VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Simulace el. obvodů pro projekt OtherKosmos

Electric Circuit Simulation for Project OtherKosmos

2020 Lukáš Kotržena

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Lukáš Kotržena**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Simulace el. obvodů pro projekt OtherKosmos
Electric Circuit Simulation for Project OtherKosmos

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je rozšířit projekt OtherKosmos o možnost simulace elektrických obvodů. V simulovaném světě bude možno vytvářet jednoduché elektrické obvody ze základních součástek jako je například baterie žárovka odpor, kondenzátor a cívka. Odděleně pak budou řešeny obvody složené z logických hradel. Projekt je zpracováván v jazyce Java a využívá knihovnu GDX pro vykreslování 3D grafického prostředí simulovaného světa.

Řešení zajistí:
1. Rozšíření simulace analogových obvodů o další součástky (např. kondenzátor, cívka).
2. Simulace logických obvodů.
3. Možnosti optimalizace simulace.
4. Začlenění simulace obvodů do simulačního modulu z projektu OtherKosmos (ukládání, nástroje, elektrárny).
5. Experimenty ověření funkcionality a výkonu.

Práce bude obsahovat:
1. Přehled použitých technologií.
2. Implementaci výše popsané funkcionality.
3. Dokumentaci programového řešení s využitím diagramů jazyka UML.

Seznam doporučené odborné literatury:

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
[2] DARWIN, Ian F. Java cookbook. 2nd ed. Sebastopol, CA: O'Reilly, c2004, xxiv, 829 p. ISBN 05-960-0701-9. Dostupné z: http://it-ebooks.info/book/2249/

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 01.09.2019

Datum odevzdání: 30.04.2020

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky 1D16 OSTRAVA

doc. Ing. Jan Platoš, Ph.D.	prof. Ing. Pavel Brandštetter, CSc.
vedoucí katedry	*děkan fakulty*

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 11. května 2020 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 11. května 2020 .....

Chtěl bych poděkovat Ing. Davidu Ježkovi, Ph.D. za skvělé vedení diplomové práce.

Abstrakt

Tato práce popisuje analýzu obvodů se stejnosměrnými i harmonickými zdroji a minimalizaci logických obvodů. Její součástí je implementace simulace elektrických a logických obvodů v jazyce Java a integrace do projektu Jiný kosmos. V herním prostředí projektu Jiný kosmos je možno stavět a interagovat s elektrickými a logickými obvody, jejichž chování je simulováno v reálném čase. Stav obvodů se ukládá a načítá z disku.

Klíčová slova: simulace elektrických obvodů; simulace logický obvodů; výuková hra

Abstract

This thesis describes analysis of circuits with direct and harmonic sources and minimization of logic circuits. It includes implementation of simulation of electrical circuits and logic circuits in Java and integration into project The Other Cosmos. It is posible to build and interact with electrical and logic circuits in the game environment of project The Other Cosmos. Electrical and logic circuits are simulated in real time. State of circuits is saved and loaded from disk.

Keywords: electrical circuit simulation; logic circuits simulation; educational game

Obsah

Seznam použitých zkratek a symbolů

JSON	–	JavaScript Object Notation
JVM	–	Java Virtual Machine
NaN	–	Not a Number

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Videohry jsou dnes velmi populární druh zábavy a některé tituly dokonce nutí hráče přemýšlet a hledat řešení nějakých problémů. Proto se videohry stále častěji objevují ve výuce. Na základních školách se například objevují hry, které vyučují základy programování. Také mezi populárními videohrami, které jsou určeny primárně pro zábavu, můžeme najít tituly, které například umožňují hráčům stavět stroje, které jsou velmi podobné logickým obvodům. Elektrické obvody se ve hrách většinou omezují jen na to, jestli je generováno dost energie, aby byla pokryta spotřeba všech zařízení.

V této práci, která navazuje na semestrální projekt, bylo cílem rozšířit projekt Jiný kosmos o simulaci elektrických a logických obvodů, umožnit stavět obvody v herním světě pomocí bloků a zajistit správné ukládání a načítání obvodů ve světě.

2 Projekt Jiný kosmos

Projekt Jiný kosmos je videohra, která je vyvíjena v jazyce Java a využívá libGDX rámec. Svět projektu Jiný kosmos je složen z voxelů a je procedurálně generován. To nabízí hráči neomezený herní prostor a možnost modifikovat celý herní svět podobně jako ve světově proslulé hře Minecraft. Dlouhodobým cílem projektu Jiný kosmos je vytvořit herně-výukové prostředí.

2.1 Počáteční stav

Na projektu Jiný kosmos již pracovalo více studentů. Neobsahuje zatím žádnou hratelnost, ale obsahuje spoustu funkčních technických prvků.

Projekt dokáže procedurálně vygenerovat terén světa a umístit do něj další struktury, jakými jsou třeba stromy nebo budovy, které jsou také procedurálně generovány. Svět se skládá z různých biotypů, které mají různé klimatické podmínky[1]. Hra dokáže efektivně vykreslit svět a zajistit kolize s hráčem a jinými objekty. Svět rozděluje na sloupce, které plynule ukládá nebo načítá z disku podle toho, jak se hráč ve světě pohybuje.

Obsahuje také systém simulací, který spouští simulační třídy v samostatném vlákně a plánuje jejich běh s ohledem na jejich prioritu. Simulace mají stavy, které jim umožňují spouštět se opakovaně v dalších simulačních krocích nebo se uspat na nějakou dobu. Některé implementované simulace jsou například simulace vody nebo explozí[2].

3 Analýza elektrických obvodů

Cílem analýzy elektrických obvodů je zjistit hodnoty proudu a napětí na různých místech v obvodu, která jsou pro nás zajímavá. Základem analýzy jsou Kirchhoffovy zákony, ze kterých jsou odvozeny dvě metody. Metoda uzlových napětí využívá prvního Kirchhoffova zákona, který pojednává o proudech tekoucích uzlem a umožňuje sestavení rovnic pro jednotlivé uzly obvodu. Metoda smyčkových proudů sestavuje rovnice proudových smyček obvodu podle druhého Kirchhoffova zákona. Výstupem obou metod je soustava lineárních rovnic, pokud je obvod složen z lineárních prvků. Vyřešením soustavy rovnic, která vznikla metodou uzlových napětí, jsou získány hodnoty napětí na uzlech. V případě metody smyčkových proudů jsou z vyřešené soustavy rovnic získány velikosti proudů tekoucí smyčkami.

Pro algoritmizaci je vhodnější metoda uzlových napětí, jelikož obvod lze jednoduše reprezentovat jako graf, kde vrcholy jsou uzly a hrany jsou prvky obvodu. U metody smyčkových proudů by bylo nutné nejprve nalézt v takovém grafu cykly.

3.1 Metoda uzlových napětí

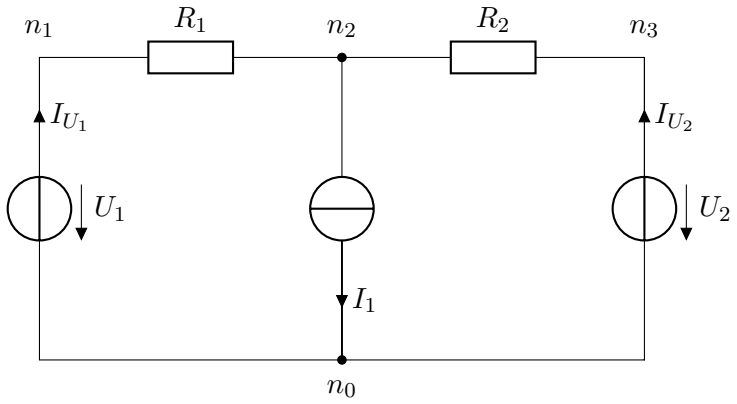
Při řešení obvodů touto metodou je nejprve potřeba zvolit referenční uzel. Jelikož veličina napětí vyjadřuje rozdíl elektrických potenciálů mezi dvěma body, vypočtené hodnoty napětí na ostatních uzlech budou vzhledem ke zvolenému referenčnímu uzlu.

Dalším krokem je sestavení rovnice pro každý uzel kromě referenčního. Rovnice se sestaví podle prvního Kirchhoffova zákona, tedy součet všech proudů v uzlu je roven nule. Polarita proudů se liší podle směru toku z uzlu nebo do uzlu. K vyjádření proudů v rovnicích jsou obvykle potřeba napětí na uzlech, což jsou hledané proměnné.

Rovnice proudů tekoucí uzlem se sestaví jako součet proudů tekoucích prvky obvodu, které jsou připojeny na uzel. Druhá strana rovnice je nula. Proud tekoucí rezistorem se vyjádří pomocí Ohmova zákona, kde úbytek napětí na rezistoru bude rozdíl neznámých napětí na uzlech. Výsledkem může být například soustava rovnic (1), která popisuje obvod na obrázku 1, kde byl uzel n_0 zvolen jako referenční.

$$\begin{aligned} U_{n_1} &= U_1 \\ \frac{U_{n_1} - U_{n_2}}{R_1} + \frac{U_{n_3} - U_{n_2}}{R_2} - I_1 &= 0 \\ U_{n_3} &= U_2 \end{aligned} \tag{1}$$

Tato metoda se komplikuje, pokud nejsou všechny zdroje napětí připojeny jedním vývodem k referenčnímu uzlu. Tento problém lze vyřešit pomocí principu superpozice. Soustava rovnic se sestaví a vyřeší pro každý zdroj zvlášť, kde ostatní zdroje napětí nahradíme zkratem. Takové řešení ale navyšuje výpočetní složitost, pokud je v obvodu zapojen více než jeden zdroj.



Obrázek 1: Příklad elektrického obvodu

3.2 Modifikovaná metoda uzlových napětí

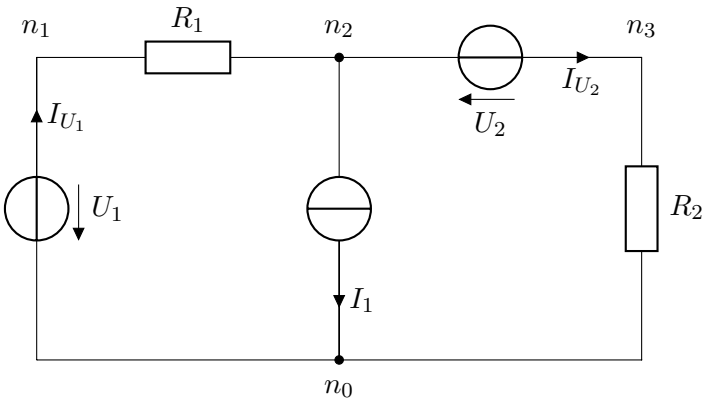
Algoritmizaci problému analýzy obvodů usnadňuje modifikovaná metoda uzlových napětí[3]. Metoda přidává rovnice pro každý zdroj napětí a umožňuje tedy řešit obvody s libovolně umístěnými zdroji. U této metody jsou navíc proudy přes napěťové zdroje jako další neznámé proměnné. Rovnice napěťových zdrojů vyjadřují velikost napětí mezi uzly, ke kterým je zdroj připojen. Rovnice uzlů se vytváří i pro uzly připojené na napěťový zdroj, zde se využije proměnná vyjadřující proud tekoucí přes zdroj.

Předpokládejme, že je dán obvod, který má označeny všechny potřebné veličiny. Postup modifikované metody uzlových napětí je následující:

- Vybrat referenční uzel.
- Sestavit proudové rovnice pro každý uzel kromě referenčního podle prvního Kirchhoffova zákona.
- Sestavit rovnice napěťových zdrojů jako rozdíl napětí mezi připojenými uzly.

Pokud je tento postup aplikován na obvod na obrázku 2 a je zvolen uzel n_0 jako referenční, výsledkem je soustava rovnic (2). V této soustavě figurují napětí na uzlech (n_1, n_2, n_3) a proudy procházející napěťovými zdroji (I_{U_1}, I_{U_2}) jako neznámé proměnné.

$$\begin{aligned}
n_1: &\quad I_{U_1} + \frac{U_{n_2} - U_{n_1}}{R_1} = 0 \\
n_2: &\quad \frac{U_{n_1} - U_{n_2}}{R_1} - I_{U_2} - I_1 = 0 \\
n_3: &\quad I_{U_2} - \frac{U_{n_3}}{R_2} = 0 \\
U_1: &\quad U_{n_1} = U_1 \\
U_2: &\quad U_{n_3} - U_{n_2} = U_2
\end{aligned} \tag{2}$$



Obrázek 2: Příklad elektrického obvodu se zdrojem, který není připojen k referenčnímu uzlu

3.3 Harmonické zdroje a modifikovaná metoda uzlových napětí

Modifikovanou metodu uzlových napětí lze jednoduše upravit, aby s ní bylo možné analyzovat obvody složené z harmonických zdrojů a prvků s reaktancí. Stačí vyjádřit impedanci prvků pomocí komplexních čísel. Postup sestavení rovnic je poté totožný jako v obvodech se stejnosměrnými zdroji a rezistory.

Reaktance prvků obvodu je závislá na frekvenci, proto nelze jednoduše analyzovat obvody, které obsahují více harmonických zdrojů s různou frekvencí. Tento problém lze vyřešit použitím principu superpozice. Není potřeba analyzovat obvod vždy s jedním aktivním zdrojem, ale stačí analyzovat obvod pro každou frekvenci, která se v něm vyskytuje. Všechny zdroje se stejnou frekvencí se ponechají působit v obvodu najednou a ostatní zdroje napětí se nahradí zkratem a zdroje proudu se odpojí. Výsledné hodnoty pro jednotlivé frekvence nelze sečíst, ale lze vypočítat okamžité hodnoty těchto veličin a ty již sečíst lze.

Podobně lze analyzovat smíšené obvody, které obsahují jak stejnosměrné zdroje, tak i zdroje harmonické. Analýza obvodu se provede samostatně pro harmonické zdroje a stejnosměrné zdroje. Stejnosměrná složka udává vertikální posun harmonického napětí.

3.4 Analýza přechodných jevů

Pomocí analýzy obvodu lze zkoumat ustálený stav obvodu, pokud je ale do stejnosměrného obvodu zapojen například kondenzátor, nastává přechodný jev. Počáteční ustálený stav je vybitý kondenzátor, který se chová jako zkrat a konečný ustálený stav je nabitý kondenzátor.

V samotné stejnosměrné analýze lze kondenzátor v obvodu nahradit napěťovým zdrojem s hodnotou napětí podle stavu nabití kondenzátoru. Vybitý kondenzátor bude mít nulové napětí. V konečném ustáleném stavu bude mít kondenzátor stejné napětí, jaké bylo mezi uzly, než byl na ně kondenzátor připojen.

Aby bylo možné zkoumat stav mezi těmito dvěma stavy, je potřeba vyjádřit stav nabití kondenzátoru v závislosti na čase. Rychlost nabíjení kondenzátoru je přímo úměrná proudu pro-

tékajícím kondenzátorem. Proud procházející kondenzátorem se mění spolu se stavem nabití kondenzátoru. Řešení lze tedy nalézt vyřešení diferenciální rovnice[4]. Závislost napětí kondenzátoru na čase, který je v jednoduché smyčce s napěťovým zdrojem a rezistorem, vyjadřuje vztah (3).

$$U_c(t) = U_0(1 - e^{-\frac{t}{RC}}) \tag{3}$$

4 Minimalizace logických funkcí

Výstup kombinatorických obvodů lze vyjádřit jako logickou funkci pomocí Booleovy algebry. V sekvenčních obvodech jsou výstupy závislé také kromě vstupů i na stavech hradel, protože se v těchto obvodech vyskytují prvky obsahující paměť, nebo jsou prvky zapojeny se zpětnou vazbou, která slouží jako paměť. Výstupy sekvenčních obvodů tedy nelze převést na logické funkce.

Logické funkce lze optimalizovat, tak aby obsahovaly co nejméně operací a tak lze zpětně zjednodušit logický obvod. Nejznámější metodou minimalizace logických funkcí je metoda Karnaughových map. Tato metoda je vhodná pro manuální minimalizaci. Metoda Quine–McCluskey je také metoda pro minimalizaci logických funkcí, která je ekvivalentní mětodě Karnaughových map, ale je vhodnější pro implementaci a umožňuje minimalizaci funkcí s velkým počtem proměnných[5].

4.1 Metoda Quine–McCluskey

Prvním krokem této metody je nalezení mintermů vstupní funkce, pokud takto není vstupní funkce již zadaná. Mintermy se vyjádří pomocí binárního čísla, tak jak by byly zapsány příslušné řádky mintermů v pravdivostní tabulce. Dále jsou mintermy rozděleny do skupin podle počtu jedniček v binárním čísle. Takto vzniknou počáteční implikanty a jejich rozdělení do skupin.

V sousedních skupinách, tedy skupinách implikantů, jejichž počet jedniček se liší pouze o jednu, lze najít implikanty, které jsou stejné kromě jedné pozice v binárním čísle. Tyto implikanty lze zkombinovat a na pozici, ve které se implikanty liší, zapsat pomlčku, která zastupuje obě hodnoty 0 a 1. Takto se naleznou kombinace implikantů mezi všemi dvojicemi sousedních skupin. Kombinované implikanty se opět zařazují do skupin, podle toho ze kterých dvou skupin pocházely původní implikanty. Nové skupiny musí zachovat stejné pořadí jako původní skupiny, ze kterých vznikly. Rozdělení do skupin snižuje počet implikantů, které je potřeba porovnat, protože se implikanty musí lišit pouze v jednom znaku a k tomu může dojít pouze v sousedních skupinách.

Na nově vzniklé skupiny implikantů, lze aplikovat stejný postup jako na počáteční skupiny. Jediný rozdíl oproti předchozímu postupu je, že žádný ze znaků, ve kterých se implikanty liší, nesmí být pomlčka. Tento postup se provádí opakovaně dokud nelze zkombinovat žádné implikanty.

Během všech iterací je potřeba označit implikanty, které byly použity ke kombinaci, respektive zaznamenat si implikanty, které nebyly použity. Nepoužité implikanty jsou tzv. primární a právě ty jsou touto metodou hledány. V posledním kroku, kdy nešly žádné implikanty zkombinovat, jsou všechny primární. Během kombinace implikantů může vzniknout duplikátní implikant, který již vznikl kombinací jiných implikantů. Takové implikanty se odstraní, aby nebyly žádné duplikáty.

Z primárních implikantů je potřeba vybrat podmnožinu implikantů, které se budou vyskytovat v minimální funkci. Implikanty musí být do podmnožiny vybrány tak, aby pokrývaly všechny mintermy původní funkce. Implikant pokrývá minterm, pokud se s jeho binární reprezentací liší

jen v pomlčkách. V minimální funkci se určitě budou vyskytovat nezbytné implikanty. Nezbytné implikanty jsou takové, které jako jediné pokrývají některý minterm, respektive pokud je minterm pokryt pouze jedním implikantem, je tento implikant nezbytný[6].

Výběrem nezbytných implikantů se vetšinou pokryjí všechny mintermy. V případě, že by nebyly pokryty všechny mintermy nezbytnými implikanty, je potřeba vybrat jinou metodou minimální podmnožinu ze zbývajících implikantů, která by pokrývala zbývající mintermy. K tomu lze použít Petrickovu metodu.

4.1.1 Příklad

Je dána logická funkce pravdivostní tabulkou 1a. Nejprve je potřeba vybrat z pravdivostní tabulky řádky odpovídající mintermům funkce a rozdělit je do skupin podle počtu jedniček. Vznikne tabulka původních implikantů 1b, kde horizontální čára odděluje skupiny implikantů. V prvním sloupci tabulky implikantů jsou zapsány dekadické podoby mintermů, ze kterých implikant vznikl.

n	**A**	**B**	**C**	**f(A, B, C)**
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

(a) Pravdivostní tabulka

	A	**B**	**C**
1	0	0	1
2	0	1	0
4	1	0	0
3	0	1	1
6	1	1	0
7	1	1	1

(b) Tabulka implikantů vzniklých z mintermů funkce

Tabulka 1: Tabulky ukázkové funkce

Nyní se v tabulce implikantů hledají implikanty, které je možné zkombinovat. Mezi první a druhou skupinou lze zkombinovat například implikanty 1 a 3. Ty se liší pouze na pozici proměnné B a kombinací vznikne první implikant v tabulce nových implikantů 2b. V tabulce předchozích implikantů 2a se v posledním sloupci značí, které implikanty byly použity ke kombinaci a v tabulce 2b je tedy implikant, který nahrazuje ty označené. V první iteraci se nesnížil počet implikantů, ale snížil se počet skupin. V každém kroku se musí počet skupin snížit nejméně o jednu.

Dále se opět hledají implikanty ke kombinaci, tentokrát z tabulky 2b. Kombinovat lze pouze implikanty, které mají pomlčky na stejných pozicích. Hned u prvního implikantu (1, 3) lze vidět, že jej nelze zkombinovat s žádným jiným implikantem. Jedná se tedy o primární implikant. Implikanty (2, 3) a (6, 7) lze zkombinovat do implikatu „–1–“. Implikanty (2, 6) a (3, 7) lze také zkombinovat do stejného implikantu „–1–“. Do tabulky nových implikantů 2c se zapíše jen

jednou, ale v tabulce předchozích implikantů 2b se musí označit obě dvojice, ze kterých může vzniknout.

	A	**B**	**C**	
1	0	0	1	✓
2	0	1	0	✓
4	1	0	0	✓
3	0	1	1	✓
6	1	1	0	✓
7	1	1	1	✓

(a) V první iteraci

	A	**B**	**C**	
1, 3	0	–	1	
2, 3	0	1	–	✓
2, 6	–	1	0	✓
4, 6	1	–	0	
3, 7	–	1	1	✓
6, 7	1	1	–	✓

(b) V druhé iteraci

	A	**B**	**C**	
2, 3, 6, 7	–	1	–	

(c) V třetí iteraci

Tabulka 2: Tabulky implikantů ukázkové funkce

Žádné další implikanty nelze kombinovat. Výstupem jsou primární implikanty, ty se vyberou z tabulek všech iterací a zapíšou do nové tabulky, kde je jednodušší najít nezbytné primární implikanty. Tabulka 3 primárních implikantů obsahuje v levé části nalezené primární implikanty a v dalších sloupcích je symbol „x“ pokud implikant pokrývá minterm, který je uveden v dekadické podobě v hlavičce tabulky.

A	**B**	**C**	**1**	**2**	**3**	**4**	**6**	**7**
–	1	–		Ⓧ	x		x	Ⓧ
0	–	1	Ⓧ		x			
1	–	0				Ⓧ	x	

Tabulka 3: Primární implikanty ukázkové funkce

Nezbytné primární implikanty lze nalézt tak, že se najdou sloupce, které obsahují pouze jeden symbol „x“. Nezbytný implikant je pak na tom řádku, na kterém se symbol „x“ vyskytuje. V tabulce 3 jsou kruhem označeny pozice, kde minterm je pokryt pouze jedním implikantem. Všechny tři primární implikanty jsou tedy nezbytné. Z nezbytných implikantů se sestaví minimalizovaná logická funkce, která má tuto podobu: $f(A, B, C) = B + \overline{A}C + A\overline{C}$.

4.2 Petrickova metoda

Pokud v metodě Quine–McCluskey nepokrývají nezbytné implikanty všechny mintermy, lze na zbývající implikanty aplikovat Petrickovu metodu. Ta spočívá v tom, že se sestaví Booleova funkce, která je pravdivá, pokud jsou pokryty všechny mintermy.

Každý zbývající primární implikant se označí a jejich značení bude použito jako proměnné ve funkci vyjadřující pokrytí všech mintermů. Funkce se sestaví tak, že pro každý minterm se vytvoří výraz, který obsahuje disjunkci primárních implikantů pokrývajících daný minterm. Tyto výrazy jednotlivých mintermů se spojí konjunkcí, a vzniká tedy funkce v konjunktivní normální formě.

V této funkci hledáme takové ohodnocení proměnných, aby hodnota funkce byla pravdivá a zároveň co nejméně proměnných bylo ohodnocených hodnotou 1. Takové ohodnocení lze nalézt převedením funkce do disjunktivní normální formy a vybráním jednoho člena disjunktivní normální formy, který obsahuje nejméně proměnných. Tyto proměnné z vybraného člena reprezentují primární implikanty, které minimálně pokrývají zbývající mintermy. [7]

4.2.1 Příklad

Aplikací Quineovy–McCluskeho metody na logickou funkci $f(A,B,C) = \sum m(0,2,3,4,5,7)$ vznikne tabulka 4, která obsahuje primární implikanty dané logické funkce. V tabulce nebyly nalezeny žádné nezbytné primární implikanty a pro nalezení minimální funkce lze použít Petrickovu metodu.

	A	**B**	**C**	**0**	**2**	**3**	**4**	**5**	**7**
P_1	0	–	0	x	x				
P_2	–	0	0	x			x		
P_3	0	1	–		x	x			
P_4	1	0	–				x	x	
P_5	–	1	1			x			x
P_6	1	–	1					x	x

Tabulka 4: Primární implikanty funkce $f(A,B,C) = \sum m(0,2,3,4,5,7)$

Každý implikant se označí a sestaví se logická funkce (4). Každá závorka vyjadřuje, které implikanty musí být vybrány, aby byl pokryt jeden minterm. Závorky lze jednoduše sestavit podle sloupců tabulky 4.

$$P = (P_1 + P_2)(P_1 + P_3)(P_3 + P_5)(P_2 + P_4)(P_4 + P_6)(P_5 + P_6) \tag{4}$$

Tuto rovnici je potřeba převést do disjunktní normální formy. To lze jednoduše provést opakováním dvou kroků: roznásobení závorek a zjednodušení pomocí pravidla absorpce $A + AB = A$. Celý postup převodu je znázorněn níže (5), kde se v jednom kroku roznásobí sousední závorky a dalším kroku se zjednodušují výrazy pomocí pravidla absorpce.

$$\begin{aligned}
P &= (P_1 + P_1P_3 + P_1P_2 + P_2P_3)(P_2P_3 + P_3P_4 + P_2P_5 + P_4P_5)(P_4P_5 + P_4P_6 + P_5P_6 + P_6) \\
&= (P_1 + P_2P_3)(P_2P_3 + P_3P_4 + P_2P_5 + P_4P_5)(P_4P_5 + P_6) \\
&= (P_1P_2P_3 + P_1P_3P_4 + P_1P_2P_5 + P_1P_4P_5 + P_2P_3 + P_2P_3P_4 + P_2P_3P_5 + P_2P_3P_4P_5)(P_4P_5 + P_6) \\
&= (P_1P_3P_4 + P_1P_2P_5 + P_1P_4P_5 + P_2P_3)(P_4P_5 + P_6) \\
&= P_1P_3P_4P_5 + P_1P_3P_4P_6 + P_1P_2P_4P_5 + P_1P_2P_5P_6 + P_1P_4P_5 + P_1P_4P_5P_6 + P_2P_3P_4P_5 + P_2P_3P_6 \\
&= P_1P_3P_4P_6 + P_1P_2P_5P_6 + P_1P_4P_5 + P_2P_3P_4P_5 + P_2P_3P_6
\end{aligned} \tag{5}$$

Z poslední podoby funkce, která je v disjunktní normální formě, se vybere minterm s nejméně součiny. V tomto případě to jsou dvě možnosti: $P_1P_4P_5$ a $P_2P_3P_6$. Z těchto dvou vybraných skupin implikantů lze sestavit dvě varianty hledané minimalizované funkce (6). V tomto případě jsou obě nalezené varianty opravdu minimální.

$$\begin{aligned} f(A,B,C) &= \overline{A}\,\overline{C} + A\overline{B} + BC \\ f(A,B,C) &= \overline{B}\,\overline{C} + \overline{A}B + AC \end{aligned} \tag{6}$$

5 Struktura projektu

Projekt Jiný kosmos používá nástroj pro sestavování Gradle. Hlavní Gradle projekt je složen z projektů `core`, který obsahuje hlavní implementaci, a `desktop`, který slouží jako spouštěč pro desktopové operační systémy. K těmto dvou projektům byl přidán další projekt, který obsahuje modul pro simulaci elektrických a logických obvodů.

Projekt simulace elektrických a logických obvodů je samostatně použitelný jako knihovna. Obsahuje testy jednotek, které pokrývají část funkčnosti modulů. Tento projekt také obsahuje výkonnostní testy, které jsou v projektu v samostatné množině zdrojových souborů podobně jako testy jednotek.

Výkonnostní testy používají rámec JMH[1]. Tyto testy lze spustit Gradle úlohou `jmh`. Také lze omezit, které testy se spouštějí, přidáním parametru `-Pinclude="ResistorsInSeries"`. Výsledky některých testů jsou uvedeny v sekci 6.11 a 7.4.1. Jedná se o výsledky testů, které byly spuštěny na počítači s procesorem Intel® Core™ i3-7020U, operačním systémem s jádrem Linux 5.3.0-42-generic a JVM OpenJDK 11.0.6. Po celou dobu provádění testů byla frekvence procesoru 2,3 GHz.

[1] https://openjdk.java.net/projects/code-tools/jmh/

6 Implementace modulu elektrických obvodů

Řešení elektrických obvodů bylo implementováno jako samostatně funkční modul, který nemá žádné další závislosti kromě standardní knihovny jazyka Java. Třídy modulu umožňují sestavit obvod, provést analýzu obvodu, krokové aktualizace a obvod jednoduše serializovat a deserializovat.

6.1 Datová struktura

Obvod lze reprezentovat neorientovaným grafem, který vzniká spojováním vývodů součástek s uzly. Uzly lze spojit s vývody součástek, ale ne s jinými uzly. Taktéž nelze spojit dva vývody součástek a vývod součástky může být připojen nejvýše k jednomu uzlu. Vývod součástky vždy patří pouze k jedné součástce.

Taková struktura zanedbává vodiče, které mají v idealizovaných obvodech nulový odpor, tudíž nejsou důležité pro analýzu obvodu. Pokud chceme zkoumat obvod, ve kterém vodiče mají nenulový odpor, stačí do obvodu zapojit rezistory představující odpor vodičů.

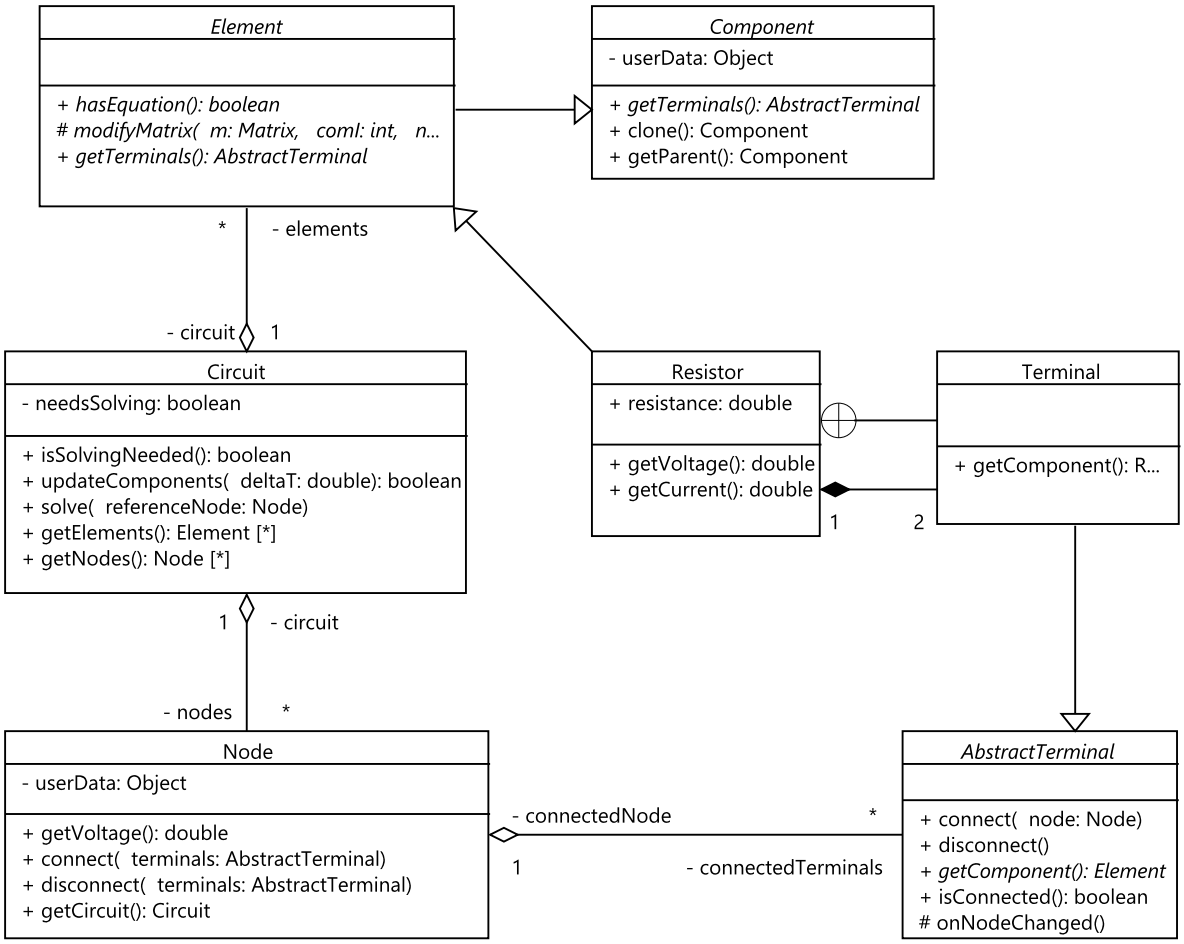
Obvod se vytváří propojováním vývodů součástek s uzly. Připojováním nebo odpojováním součástek se vytváří nebo aktualizuje instance třídy `Circuit`, která obsahuje reference na všechny součástky a uzly, které patří do stejného obvodu. Obvodem se rozumí jedna celá komponenta grafu propojení, tedy samostatně řešitelný obvod. Spojením součástek a uzlů z různých obvodů se jednoduše sjednotí jejich obvody. Rozpojení součástky od uzlu nejdříve smaže reference na obvod všech prvků obvodu a začne prvky procházet z odpojeného uzlu a přiřazovat jim nový obvod. Pokud se nový obvod nepřiřadil i odpojené součástce, pak se projdou prvky od odpojené součástky a přiřadí se jim jiný nový obvod.

Na obrázku 3 je třídní UML diagram znázorňující vztahy hlavních tříd a třídy rezistoru.

6.2 Implementace analýzy obvodu

Z jakékoliv součástky nebo uzlu lze získat referenci na obvod, do kterého patří, pomocí metody `getCircuit`. Obvod vyřešíme zavoláním metody `solve` na objektu obvodu. Nejprve se vybere referenční uzel, pokud nebyl žádný zadán. Vhodná volba referenčního uzlu jsou uzly připojené na záporné vývody zdrojů napětí nebo proudu. Výsledek analýzy obvodu je nezávislý na zvoleném referenčním uzlu, proto je zvolen poslední uzel některé ze součástek, které přidávají do soustavy rovnici. Tím je odstraněna závislost analýzy obvodu na konkrétních součástkách. Následně se sestaví matice reprezentující soustavu rovnic. Matice je čtvercová a její velikost je rovna součtu počtu uzlů bez referenčního uzlu a počtu součástek, které přidávají do soustavy další rovnici.

Prvky matice jsou vyjádřeny čísly s pohyblivou řádovou čárkou. K matici je přidán vektor pravých stran rovnic, jehož prvky jsou vyjádřeny jako lineární výraz pomocí třídy `LinearExpression`. Ta uchovává lineární výraz v podobě součtu konstanty a proměnných, které mohou být násobeny konstantou. Třída umožňuje provádět aritmetické operace nad lineárními



Obrázek 3: Třídní diagram modulu pro řešení obvodů

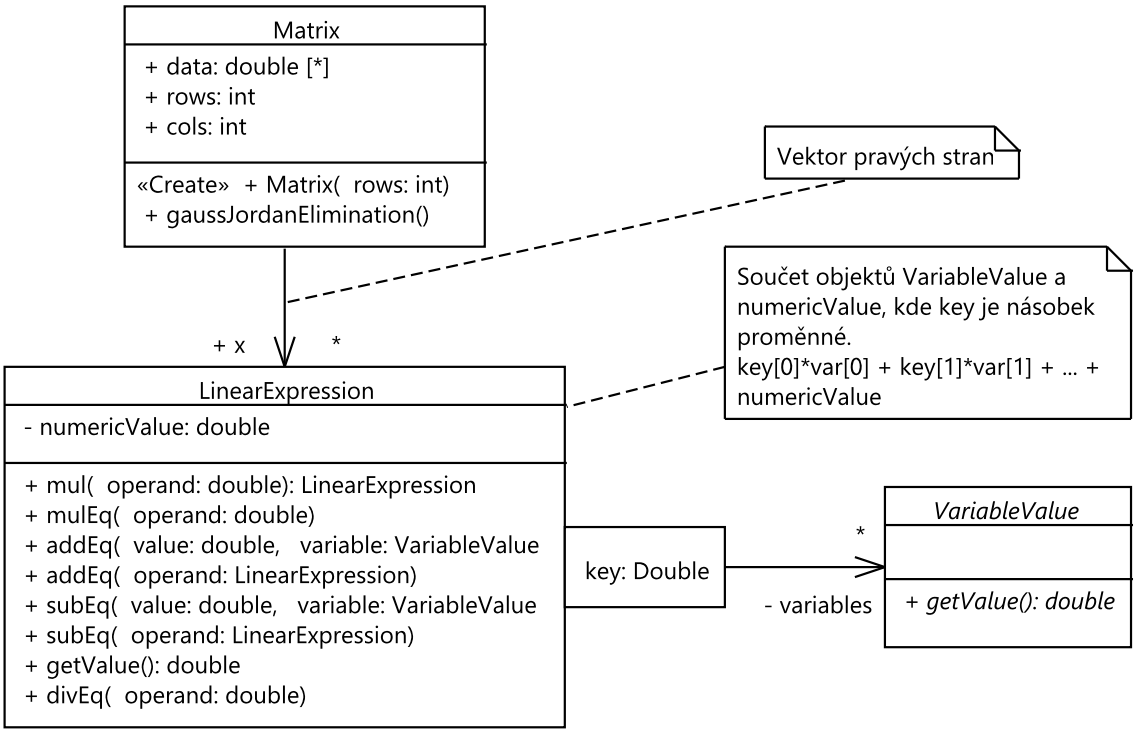
výrazy, ale neumožňuje násobení nebo dělení s jinými aritmetickými výrazy. Tyto pomocné matematické třídy jsou vyobrazeny na obrázku 4.

Vyplnění matice probíhá tak, že se předá každé součástce v obvodu v metodě `modifyMatrix`. Pořadí rovnic v matici je předem určeno. Nejprve jsou v matici řádky pro rovnice proudů v uzlu a pak řádky rovnic součástek. Spolu s maticí jsou předány seznamy uzlů a součástek, které určují pořadí rovnic. Součástky do matice doplní jim relevantní části. Rezistory sice nepřidávají do soustavy vlastní rovnice, ale do rovnic uzlů, ke kterým jsou připojeny, přidávají proud jimi protékající.

Na matici je aplikována Gaussova-Jordanova eliminační metoda. Výsledkem jsou výrazy na pravých stranách, které vyjadřují hodnotu napětí na uzlech nebo jinou veličinu v případě rovnic, které byly přidány součástkami.

Řešení pravých stran rovnice v podobě lineárního výrazu umožňuje přeskočit analýzu obvodu v případě, že se změní pouze hodnota proměnné na pravé straně soustavy rovnic. Pravé strany rovnic v modifikované metodě uzlových napětí obsahují pouze konstantní hodnoty, které nejsou

závislé na proměnných soustavy rovnic. Jedná se tedy o parametry soustavy rovnic. To jsou například hodnoty napětí na napěťových zdrojích a jejich skutečná hodnota nemá vliv na řešení soustavy rovnic.



Obrázek 4: Třídní diagram pomocných tříd pro řešení soustavy obvodu

6.3 Implementace spínačů

Přepínače a spínače nejsou v analýze elektrických obvodů řešeny a jsou rovnou interpretovány jako spojené nebo rozpojené vodiče. V simulátoru elektrických obvodů jsou implementovány, protože slouží jako vstup uživatele a umožňují interakci s obvodem.

Spínače lze přidat do soustavy rovnic obvodu a to jako napěťový zdroj s nulovým napětím, pokud je spínač sepnutý. To by umožnilo přidat nebo odebrat pouze tuto jednu rovnici z počáteční soustavy při změně stavu spínače, ale pořád je potřeba provést řešení soustavy. Každý sepnutý spínač navyšuje složitost soustavy rovnic, ale zásadní problém nastává, pokud rozepnuté spínače rozdělují obvod na více nepropojených částí. Soustava rovnic takového obvodu může být neřešitelná a obvod porušuje pravidlo zavedené v sekci 6.1, že instance třídy `Circuit` představuje samostatně řešitelný obvod.

Spínač tedy může být najednou ve dvou samostatně řešitelných obvodech. Proto bylo zavedeno omezení, že obvod může obsahovat pouze elementární součástky (instance třídy `Element`), které jsou součástí jen jednoho obvodu. Spínač je tedy implementován jako složená součástka,

která se skládá ze dvou elementárních součástek s jedním vývodem. Tyto elementární součástky slouží jen k připojení spínače a v soustavě rovnic obvodu neprovádějí žádnou změnu. Spínač dědí z obecnějšího typu `Component` a jeho vývody jsou typu `DummyElement`.

Sepnutí spínače by tedy mohlo sloučit uzly na obou stranách součástky, ale při rozepnutí by nebylo jednoznačné, jak rozdělit připojené vývody součástek do dvou uzlů. Z tohoto důvodu je možno spojovat uzly do skupin pomocí objektů typu `MultiNode`. Při analýze obvodu se se sloučenými uzly zachází jako s jedním uzlem. Pokud by bylo k jednomu uzlu připojeno více spínačů, bylo by rozdělování navrstvených skupin uzlů zbytečně složité. Aby nevznikaly složité struktury seskupených uzlů, `MultiNode` může obsahovat je elementární uzly. Spínače při sepnutí seskupí uzly, pokud nad uzly již nějaká skupina neexistuje. Pokud má spínač seskupit uzly, které jsou již ve skupinách, přidá uzel do skupiny nebo sjednotí jednotlivé skupiny. Když je spínač rozepnut, rozpustí se celá skupina a ostatní spínače, připojené k daným uzlům, budou upozorněny o změně zapojení obvodu a znovu seskupí potřebné uzly.

Vytvoření nebo rozpuštění skupiny uzlů může způsobit rozdělení nebo sloučení obvodů, ale toto řešení nezpůsobuje navýšení složitosti soustavy rovnic.

Vícecestné přepínače lze jednoduše složit z více jednocestných spínačů. Také je lze jednoduše implementovat jako samostatnou součástku přidáním dalších vývodů. V projektu je implementován například jednopólový dvoucestný přepínač třídou `SPDTSwitch`.

6.4 Implementace logických prvků

Pomocí simulace elektrických obvodů lze snadno simulovat logické obvody, pokud přidáme logické prvky jako součástky. Chování logických bran lze napodobit spojením jednopólového dvoucestného přepínače a dalších dvou vstupních vývodů. Porovnáním hodnoty napětí na vstupu s připojenou zemí lze určit logickou hodnotu vstupu. Podle logické funkce brány se nastaví stav přepínače, který přepíná výstup brány mezi napájením a zemí.

Logické brány vyžadují zapojení většího počtu vývodů, ale výsledné matice pro řešení obvodů sestaveného z logických bran je velmi malá, protože se v běžných logických obvodech vyskytují pouze dva uzly poté, co jsou vytvořeny skupiny uzlů. Nevýhodou je, že se musí obvod řešit vícekrát, pokud je zapojeno více logických bran za sebou. Opakované řešení obvodů je také potřeba pokud logický obvod obsahuje zpětnou vazbu.

Logické brány implementují rozhraní `ICircuitSolvedListener`, to jim umožní aktualizovat stav svého výstupu po vyřešení obvodu. Změna stavu výstupu logické brány označí obvod jako nevyřešený a je potřeba znovu spustit řešení obvodu. V některých případech může změna stavu výstupu brány způsobit rozdělení obvodu.

Od toho řešení bylo upuštěno a bylo nahrazeno samostatným řešením logických obvodů, které je popsáno v sekci 7.

6.5 CircuitContext

Změna stavu přepínačů může rozpojovat obvody a může se tedy stát, že po zavolání metody pro vyřešení obvodu bude obvod prázdný. To může nastat, pokud řešení obvodu způsobí přepnutí přepínačů, které rozdělí obvod. Tím vzniknou nové instance třídy `Circuit`. Takto se ztratí reference na obvod a nová se musí získat zavoláním metody `getCircuit` na některý prvek obvodu.

Tento problém byl původně řešen tak, že metoda `solve` vracela množinu všech nově vzniklých obvodů. Tím se komplikuje opakované řešení obvodu, protože uživatel musí udržovat množinu obvodů a na všech spouštět řešení obvodu.

Proto byla zavedena další vrstva, která seskupuje všechny obvody, které spolu mohou nějak interagovat. Tím je třída `CircuitContext`, která umožňuje pracovat s více obvody najednou. Obsahuje množinu obvodů, které jsou propojeny jakoukoliv součástkou typu `Component`, tím může být i rozpojený spínač. Oproti tomu třída `Circuit` obsahuje součástky a uzly, které jsou propojeny pouze elementárními součástkami typu `Element`.

Pokud se pouze spouští řešení obvodu nebo aktualizace součástek, je zaručeno, že instance kontextu obvodu zůstane stejná a bude stále obsahovat stejné součástky a uzly, které mohou být jen seskupeny v jiných obvodech. To už není zaručeno, pokud je změněno zapojení obvodu uživatelem.

Na obrázku 5 je opět třídní diagram, ale tentokrát s třídami `MultiNode`, `CircuitContext` a `DummyElement`.

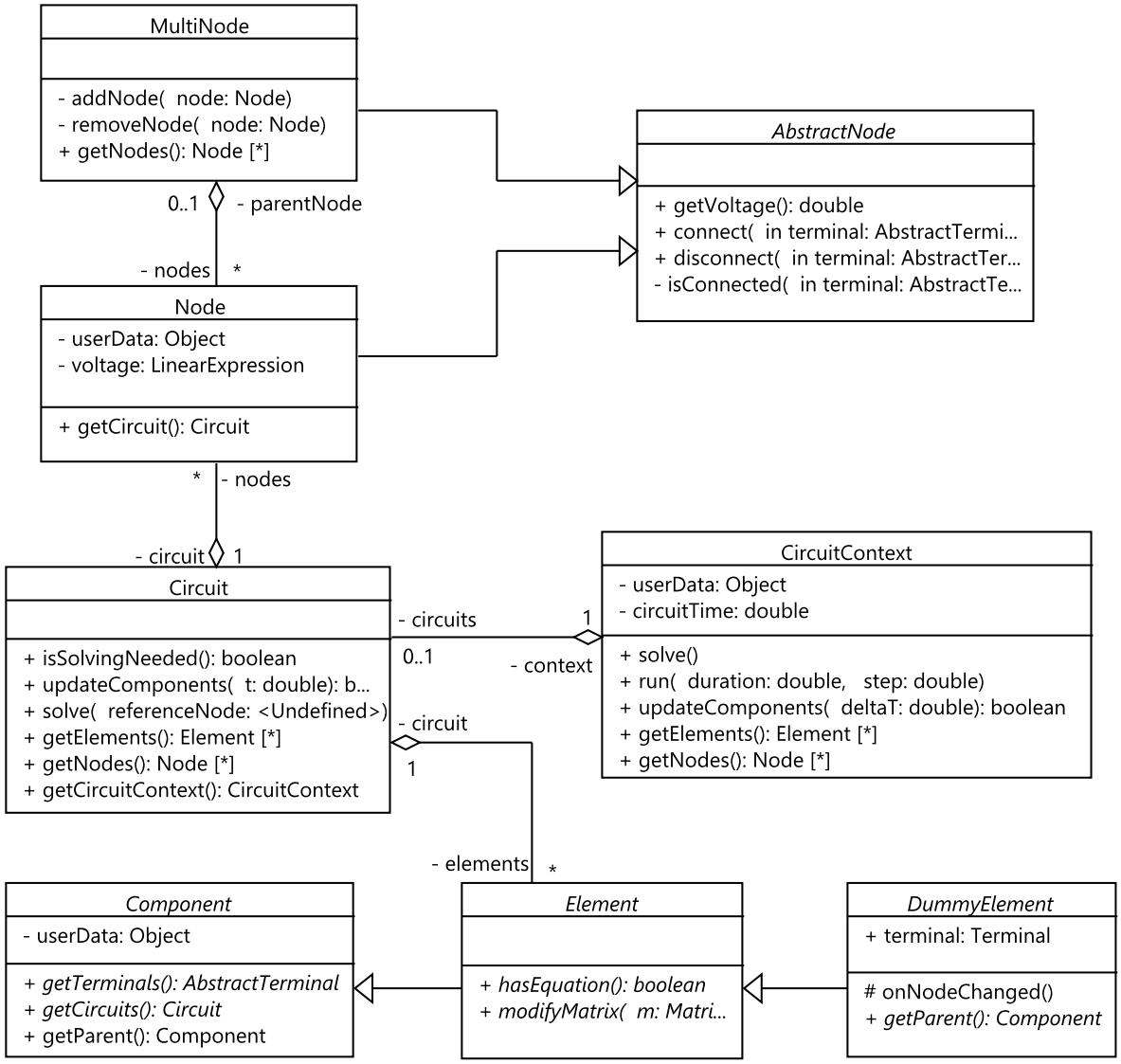
6.6 Krokové aktualizace

Některé součástky mohou vyžadovat krokové aktualizace. Třída obvodu má metodu `updateComponents` s parametrem udávajícím délku kroku. Zavoláním této metody se spustí aktualizace na všech součástkách obvodu, které implementují rozhraní `IStepUpdate`. Krokové aktualizace jsou potřeba pro aktualizaci součástek, které způsobují v obvodu přechodný jev.

Aktualizace součástek se spouští jednotlivě a pro právě aktualizovanou součástku se jeví zbytek veličin obvodu jako konstantní. To může zanášet chybu do simulace obvodu, jejíž velikost je závislá na velikosti kroku. Vhodnějším řešením by bylo řešit aktualizace součástek jako soustavu diferenciálních rovnic.

6.7 Zdroje napětí a proudu

Jak již bylo zmíněno v sekci 3.2, napěťové zdroje přidávají do soustavy rovnici, která vyjadřuje rozdíl napětí mezi uzly a přidává proud zdrojem jako neznámou proměnnou. Problém může nastat, pokud je napěťový zdroj zapojen do zkratu. To způsobí, že soustava rovnic je neřešitelná, protože proud takto zapojeným zdrojem by měl být nekonečný. Proto byla přidána možnost nastavit zdrojům vnitřní odpor, který omezuje tento proud na konečnou hodnotu. Vnitřní odpor není třeba přidávat do obvodu jako samostatnou součástku, to by přidalo do obvodu uzel, který



Obrázek 5: Třídní diagram modulu pro řešení obvodů s třídou `CircuitContext`

by byl mezi rezistorem vnitřního odporu a zdrojem napětí. Místo toho je úbytek napětí na imaginárním rezistoru odečten od napětí zdroje. Toto lze snadno zapsat, protože se v soustavě vyskytuje proměnná vyjadřující proud zdrojem (7).

Proudové zdroje nepřidávají rovnici do soustavy rovnic. Stačí totiž přidat na pravé strany rovnic uzlů hodnotu proudu zdroje s polaritou podle toho jestli do uzlu vtéká nebo z něj vytéká. Opět ale vzniká problém, pokud je zdroj nesprávně zapojen. Tentokrát v případě, kdy je proudový zdroj zapojen tak, že netvoří proudovou smyčku. V takovém případě proudový zdroj vygeneruje nekonečné napětí. Řešením je opět přidání vnitřního odporu, ten je u napěťových zdrojů v sérii, ale u proudového zdroje by sériově zapojený odpor nepomohl. Proto je vnitřní odpor proudového zdroje zapojen paralelně a tím omezí napětí, které může proudový zdroj vy-

$$U_{n_{u+}} - U_{n_{u-}} - R_{int} I_u = U_u \tag{7}$$

kde:

U_{n_i} = napětí na uzlu i
I_u = proud napěťovým zdrojem
$u+$ = uzel připojený ke kladnému vývodu napěťového zdroje
$u-$ = uzel připojený k zápornému vývodu napěťového zdroje

generovat. Vnitřní odpor je přidán do rovnic uzlů stejně jako normální rezistor, v tomto případě totiž nemůže přidat nový uzel do obvodu ani nijak jinak zvýšit složitost obvodu.

6.7.1 Kondenzátor a cívka

Kondenzátor se chová jako napěťový zdroj, který mění své napětí, podle proudu, který jím protéká. Jedná se tedy o rozšíření napěťového zdroje s implementovaným rozhraním pro krokové aktualizace. Napětí na kondenzátoru v jednoduchém obvodu, lze jednoduše vyjádřit se závislostí na čase pomocí vztahu (3) popsaného v sekci 3.4.

Tento vztah je závislý na odporu mezi napěťovým zdrojem a kondenzátorem, to může být libovolně složitý obvod. Nelineární součástky jsou v obvodu nahrazeny lineárními a v lineárních obvodech platí Théveninova věta, která říká, že jakkoliv složitý lineární obvod lze nahradit jedním zdrojem napětí v sériovém zapojení s jedním rezistorem. Stačí tedy najít hodnotu napětí a odporu.

Kondenzátor je v analýze obvodu nahrazen napěťovým zdrojem, je tedy pro něj vypočten proud jím procházející, který je vyjádřen pomocí lineárního výrazu z hodnot napěťových a proudových zdrojů v obvodu. Díky tomu lze sestavit rovnici pro proud procházející kondenzátorem (8). Maximální napětí plně nabitého kondenzátoru je stejné jako hledané napětí zdroje zástupného obvodu. Když kondenzátor dosáhne plného nabití, proud jím procházející bude roven nule a z rovnice pro proud procházející kondenzátorem lze vyjádřit maximální napětí, kterého kondenzátor může dosáhnout (9).

$$I_c = x_c U_c + \sum_{i=1}^{n} x_i U_i \tag{8}$$

$$U_{cmax} = -\sum_{i=1}^{n} \frac{x_i}{x_c} U_i \tag{9}$$

kde:

I_c = proud procházející kondenzátorem
U_c = napětí na kondenzátoru
x_i = koeficienty
U_i = napětí ostatních zdrojů

Jelikož je znám proud procházející kondenzátorem a bylo vyjádřeno nabíjecí napětí, lze z nich jednoduše vyjádřit odpor. V jednom kroku se počítá pouze přírůstek napětí kondenzátoru, je tedy potřeba odečíst aktuální napětí od nabíjecího napětí. Vztah lze také použít pro vybíjení kondenzátoru, polarita protékajícího proudu a nabíjecího napětí bude v takovém případě opačná.

Vztah pro cívku je velmi podobný. V analýze obvodu je cívka nahrazena proudovým zdrojem a vztah (10) vyjadřuje hodnotu proudu na cívce.

Změnou stavu kondenzátoru nebo cívky se změní pouze hodnota, která se zapisuje na pravou stranu soustavy rovnic, není tedy potřeba znovu spouštět analýzu obvodu.

$$I_L(t) = I_0 \cdot \left(1 - e^{-\frac{U_L \cdot t}{I_0 \cdot L}}\right) \tag{10}$$

6.8 Baterie

Krokové aktualizace byly také použity pro vybíjení baterie, která podle odebíraného proudu sníží stav nabití a výstupní napětí. Baterie používá pro napětí vztah (11), který je použit v modelu chování baterie v softwaru Simscape[8].

$$v(s) = V_0 \left(\frac{s}{1 - \beta(1 - s)}\right) \tag{11}$$

kde:

s = stav nabití $\langle 0, 1\rangle$
V_0 = jmenovité napětí baterie
β = parametr určující tvar křivky $\langle 0, 1)$

V krokových aktualizacích je potřeba zjistit o kolik změnit stav nabití a podle něj nastavit napětí. K baterii je přidán parametr určující kapacitu baterie ve watt sekundách. Aktuální napětí a proud procházející baterií je znám a je tedy z nich možné vypočítat okamžitý elektrický výkon. Okamžitý elektrický výkon udává rychlost změny stavu nabití baterie, ale také je na stavu nabití baterie závislý. Jedná se tedy o diferenciální rovnici a v tomto případě byla použita Rungeova–Kuttova numerická metoda pro řešení diferenciálních rovnic 4. řádu (12).

$$
\begin{aligned}
p(s) &= u(s) \cdot i(s) \\
s(t+\Delta t) &= s(t) + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4) \\
K_1 &= \frac{p(s(t))}{P_c}\Delta t \\
K_2 &= \frac{p(s(t) + \frac{1}{2}K_1)}{P_c}\Delta t \\
K_3 &= \frac{p(s(t) + \frac{1}{2}K_2)}{P_c}\Delta t \\
K_4 &= \frac{p(s(t) + K_3)}{P_c}\Delta t
\end{aligned}
\tag{12}
$$

kde:

P_c = kapacita baterie

6.9 Implementace harmonických obvodů

Analýza harmonických obvodů lze provést samostatně díky principu superpozice, který platí, protože všechny součástky obvodu v stejnosměrné analýze jsou zastoupeny lineárním prvkem a harmonické obvody se chovají lineárně. Samotná analýza probíhá velmi podobně jako u stejnosměrných obvodů. Reaktance prvků obvodů je závislá na frekvenci, je tedy nutné řešit zdroje harmonického napětí s různou frekvencí samostatně.

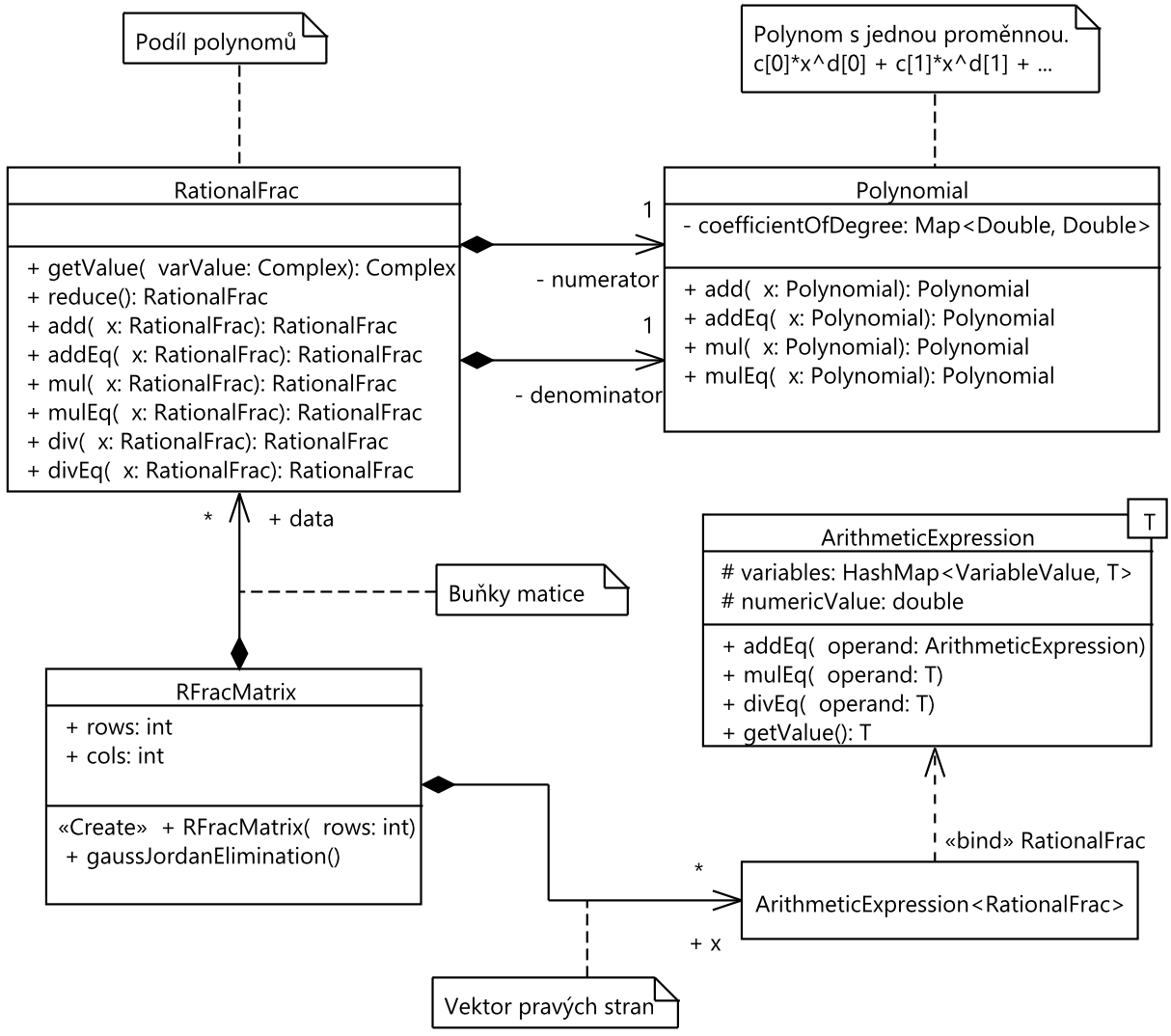
Impedance prvků obvodu jsou komplexní čísla, buňky matice soustavy by tedy musely být komplexní čísla a soustava by musela být řešena samostatně pro každou frekvenci. Pokud je vyjádřena impedance pomocí komplexní frekvence s, lze soustavu řešit bez použití komplexních čísel s s jako parametrem. Řešení soustavy rovnic s parametrem je sice složitější než řešení komplexní soustavy rovnic, ale výsledek řešení soustavy s parametrem lze aplikovat na různé frekvence. Komplexní soustavy by se musely vyřešit pro každou frekvenci samostatně.

Buňky matice obsahují součet číselných hodnot, které mohou být násobeny nebo děleny parametrem s. Pokud by buňky matice byly reprezentovány pouze polynomem, nastal by problém při dělení, protože polynomy nelze vždy vydělit beze zbytku. Proto jsou buňky reprezentovány jako podíl polynomů. Dělení podílů polynomů lze převést na násobení, kde druhý činitel bude převrácená hodnota dělitele. Výběr pivotu v Gaussově eliminační metodě preferuje buňky matice bez parametru, bez parametru ve jmenovateli nebo buňky s kratšími polynomy.

Neustálým násobením polynomů může růst jeho složitost, proto je zlomek po takových operacích zjednodušován. Kromě jednoduchého krácení se také snižují exponenty čísel s plovoucí řadovou čárkou. Najde se nejmenší a největší exponent a poté se oba polynomy zlomku vynásobí, tak aby se oba extrémy přiblížily nule. Tím se zamezí případům, kdy exponenty čísel s plovoucí řadovou čárkou dosáhnou extrému. Dalším způsobem zjednodušení zlomku je dělení polynomů,

ale pokud má výsledek dělení čitatele jmenovatelem zbytek, nemůže být výsledek dělení použit. I přes tyto pokusy o zjednodušení zlomků dochází při řešení složitějších matic s častým výskytem parametru ke vzniku příliš složitých zlomků, které způsobí vznik nekonečen a NaN hodnot.

Všechny prvky obvodu musí implementovat jak metodu `modifyMatrix`, která slouží pro vyplnění matice stejnosměrného obvodu, tak i metodu `modifyACMatrix`, která vyplňuje matici pro harmonické obvody. Každý prvek musí mít definované chování v stejnosměrných i harmonických obvodech. Třídní diagram pomocných tříd, které jsou použity pro řešení harmonických obvodů, je na obrázku 6.



Obrázek 6: Třídní diagram pomocných tříd použitých k řešení harmonických obvodů

Po vyřešení soustavy harmonického obvodu, je do uzlu uloženo napětí v podobě výrazu s komplexní frekvencí s jako parametrem, který je reprezentován instancí `ACValueExpression`. Metoda `getValue` (výpis 1) vrací napětí v podobě součtu harmonický složek napětí. Vzniká tak

```
public ACValue getValue() {
   ACValue sum = new ACValue();
   for(Entry<VariableValue, RationalFrac> item: rationalFrac) {
      if(item.getKey() instanceof ACValue) {
         ACValue var = (ACValue)item.getKey();
         for(ACComponent c: var.getACComponents()){
            sum.addACComponent(
               new ACComponent(
                  item.getValue().getValue(new Complex(0, 2*Math.PI*c.frequency)).mul(c.
                      voltage),
                  c.frequency
               )
            );
         }
      }
   }
   return sum;
}
```

Výpis 1: Sestavení hodnoty složené harmonické veličiny

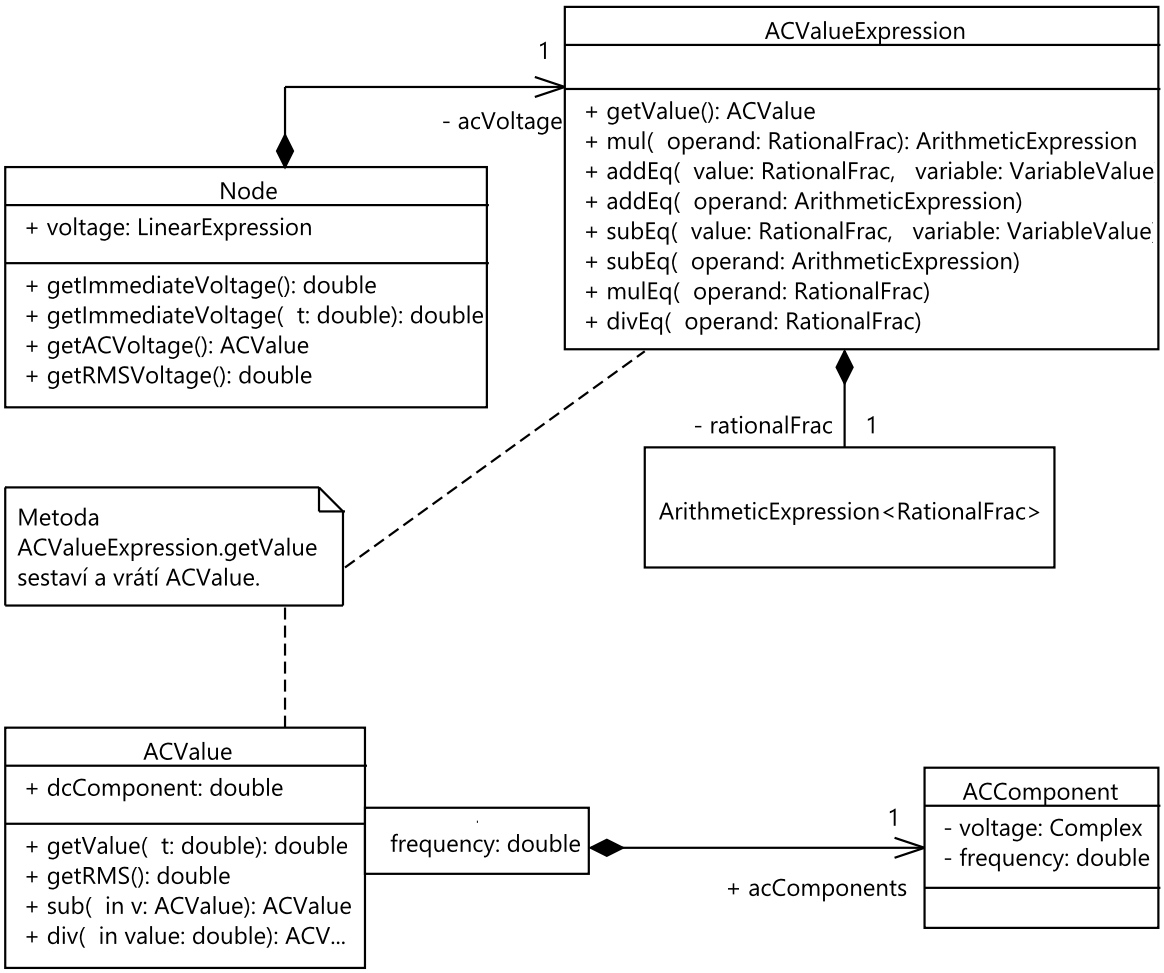
instance třídy `ACValue`, která obsahuje jednu hodnotu napětí pro jednu frekvenci vyskytující se na uzlu. Tyto harmonické složky jsou vyjádřeny komplexním číslem. Napětí vyjádřené instancí třídy `ACValue` umožňuje zjistit okamžitou hodnotu napětí a lze ho také použít k výpočtu jiných veličin, jako například proudy procházející součástkami obvodu, které budou mít také harmonický průběh. Na obrázku 7 lze vidět třídní diagram těchto tříd.

6.10 Dioda

Dioda byla implementována čistě jako usměrňovací prvek. V analýze obvodu je nahrazena napěťovým zdrojem, jehož napětí je ve výchozím stavu nulové. Dioda pomocí krokových aktualizací sleduje proud, který jí prochází. Pokud diodou protéká záporný proud, nastaví napětí zástupného zdroje tak, aby diodou netekl žádný proud.

Správná hodnota napětí, která je potřeba k zastavení záporného proudu, je vypočtena z proudu procházejícím zástupným napěťovým zdrojem diody. Proud tekoucí napěťovým zdrojem je vypočten v analýze obvodu a je tedy vyjádřen jako lineární výraz. V tomto výrazu se vyskytuje také proměnná, která vyjadřuje napětí zdroje, který zastupuje diodu. Aby diodou netekl žádný proud, musí být hodnota tohoto výrazu nula. Vzniká tak rovnice, kde neznámá proměnná je napětí zástupného zdroje diody.

V případě zapojení více diod se může stát, že diody se navzájem ovlivňují napětím svých zástupných zdrojů. Správná hodnota napětí pro zamezení toku proudu se proto musí nalézt složením rovnic diod do soustavy. Při řešení této soustavy rovnic se může stát, že mezi jejími rovnicemi je lineární závislost a existuje tedy nekonečné množství řešení. V tomto případě bude



Obrázek 7: Třídní diagram vyjádření harmonických veličin

matice soustavy obsahovat po vyřešení nulové řádky. Proměnným, jejichž výsledky měly být na nulových řádcích, bude přiřazena hodnota nula.

V analýze harmonických obvodů je dioda zastoupena rezistorem s velmi vysokým odporem, jehož účel je zablokovat harmonické napětí. Dioda má tedy různý odpor v harmonické a stejnosměrné analýze obvodu. Výchozí hodnoty odporu jsou 1 Ω v stejnosměrné analýze, aby nevznikla smyčka s nulový odporem a 100 MΩ pro zablokování harmonického napětí. Okamžitá hodnota úbytku napětí na rezistoru v harmonické analýze je použita jako napětí zástupného stejnosměrného zdroje diody s opačnou polaritou. Dále se provádí kontrola proudu diodou a zastavení záporného proudu stejně jako u stejnosměrných obvodů.

Usměrněné harmonické napětí nemá hladký průběh, jeho skoky jsou závislé na velikosti časového kroku aktualizace součástek obvodu. Při vyšších frekvencích se může stát, že není dostatek aktualizací v jedné periodě harmonického napětí a průběh usměrněného napětí pak nevypadá tak jak by se dalo očekávat. Proto byla součástkám obvodu přidána možnost doporučit

velikost časového kroku aktualizace. Dioda doporučuje krok o velikost dvacetiny periody složky harmonického napětí s nejvyšší frekvencí. Metoda provádějící simulaci obvodu zjišťuje nejmenší doporučený krok a v případě potřeby provede v rámci jedné krokové aktualizace více aktualizací součástek s menším krokem.

6.11 Výkonnostní testy

Nejnáročnější operací analýzy obvodu je vyřešení soustavy rovnic, což se provádí Gaussovou–Jordanovou metodou, jejíž asymptotická složitost je $O(n^3)$. V tabulce 5 jsou výsledky výkonnostních testů řešení obvodů složených z sériově zapojených rezistorů s jedním stejnosměrným napěťovým zdrojem. Počet uzlů v takovém obvodu je roven $n + 1$, kde n je počet rezistorů a velikost matice soustavy je $n + 1$ řádků a $n + 2$ sloupců, protože napěťový zdroj přidává do soustavy další rovnici a jeden uzel je referenční a nevyskytuje se v soustavě rovnic. V tomto případě je vynechána harmonická analýza obvodu, protože sestavená matice soustavy harmonického obvodu obsahuje ve vektoru pravých stran pouze nuly. Na obrázku 8 jsou krabicové grafy výsledků těchto výkonnostních testů, které jsou proloženy funkcí $f(x) = ax^b + c$, kde $b = 3.10297$.

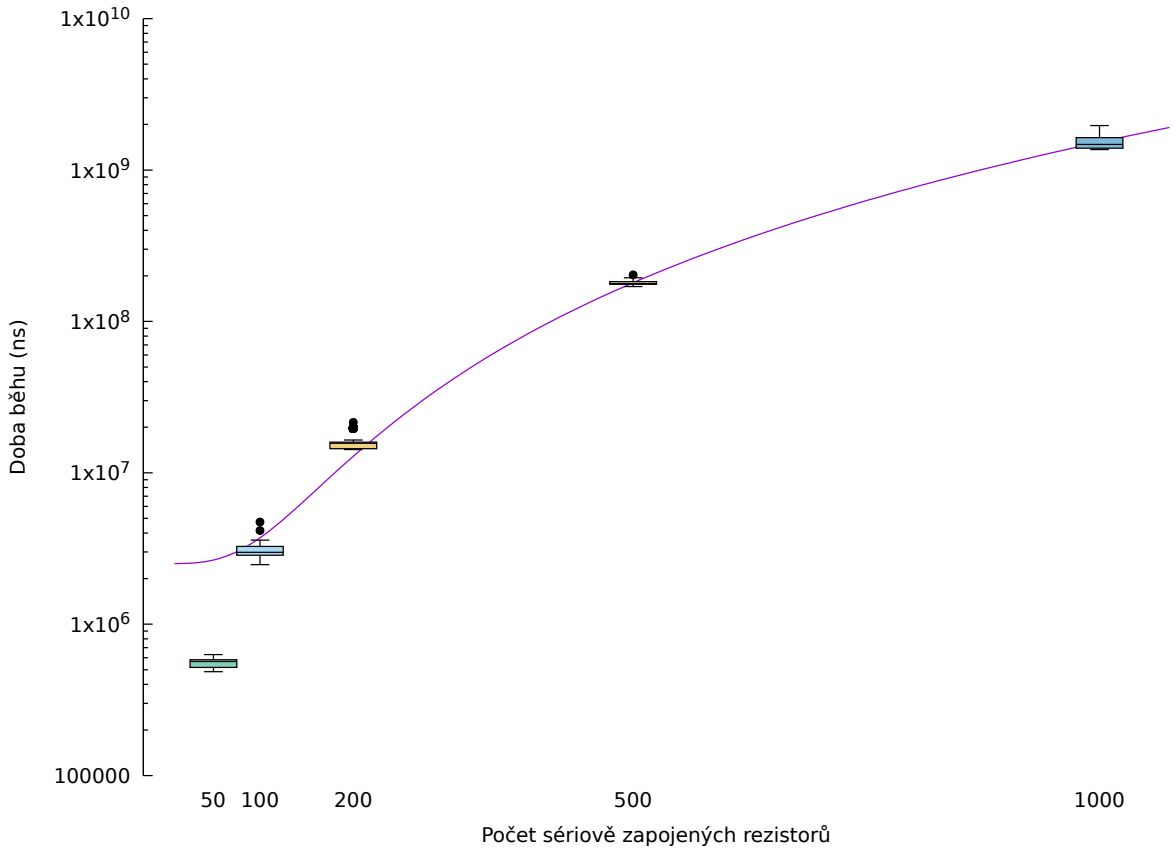
V projektu Jiný kosmos používají simulace krok o délce 50 ms. Při vyšší zátěži se tento krok může zvýšit, to ale znamená, že se také sníží frekvence aktualizace světa a to může ovlivnit herní zážitek. V tabulce 5 lze vidět, že při 200 zapojených rezistorech zabírá výpočetní doba řešení obvodu velkou část simulačního kroku. Řešení obvodu se provádí většinou pouze jednou, dopad na plynulost hry je tedy minimální. Vhodné by bylo rozšířit simulační systém projektu Jiný kosmos, aby umožnil běh takovýchto dlouhých simulací na samostatném vlákně, které by nebylo synchronizováno s aktualizací světa.

Parametry výkonnostních testů	
Zahřátí	30 iterací, každá 1 sekundu
Měření	100 iterací, každá 1 sekundu
Větvě	5
Vlákna	1
Režim	průměrný čas běhu

Výkonnostní test	**Skóre**	**Chyba**	**Jednotky**
Řešení obvodu 50 sériově zapojených rezistorů	553	± 6	μs/op
Řešení obvodu 100 sériově zapojených rezistorů	3011	± 47	μs/op
Řešení obvodu 200 sériově zapojených rezistorů	16038	± 290	μs/op
Řešení obvodu 500 sériově zapojených rezistorů	179668	± 888	μs/op
Řešení obvodu 1000 sériově zapojených rezistorů	1528128	± 21604	μs/op

Tabulka 5: Výkonnostní testy metody řešení obvodů sériově zapojených rezistorů s jedním stejnosměrným napěťovým zdrojem

Výsledky výkonnostních testů harmonických obvodů jsou v tabulce 6 a na obrázku 9. V tomto případě byla prováděna analýza stejnosměrných obvodů i harmonických obvodů, protože kondenzátory jsou ve stejnosměrné analýze nahrazeny napěťovým zdrojem a není tedy stej-



Obrázek 8: Krabicový graf výsledků výkonnostních testů metody řešení obvodů sériově zapojených rezistorů s jedním stejnosměrným zdrojem

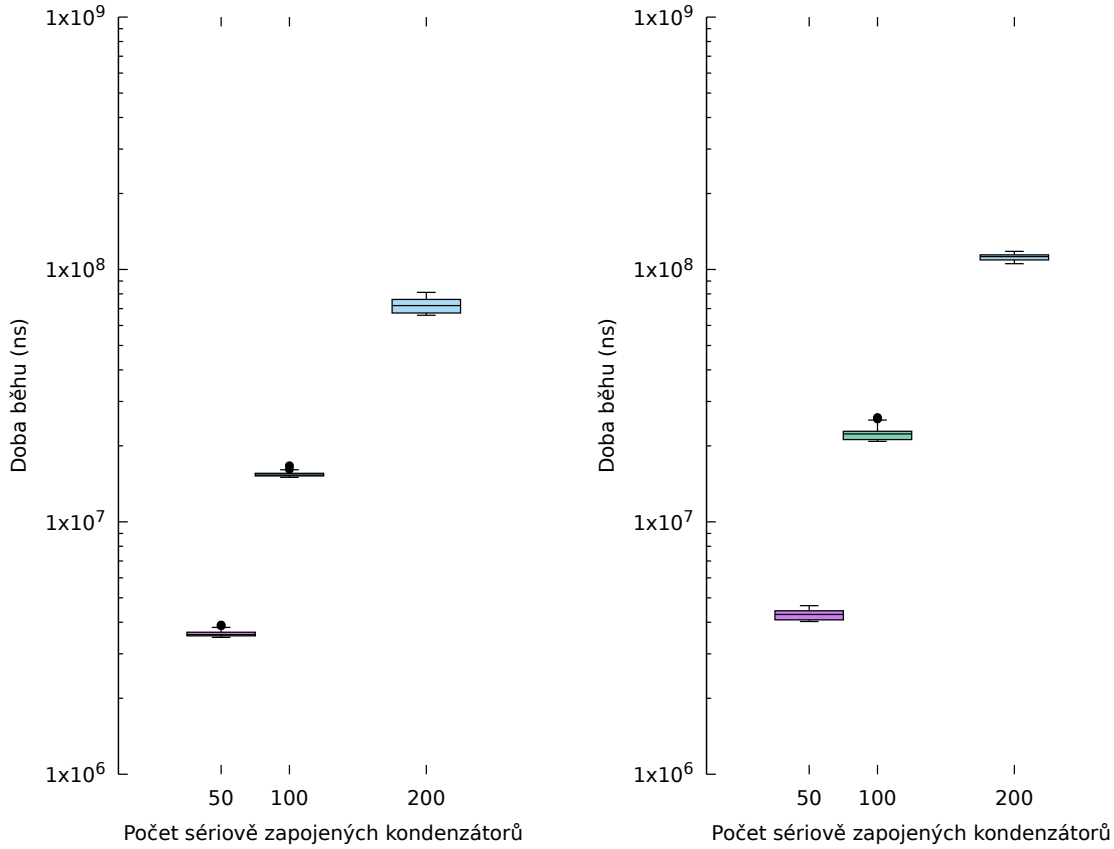
nosměrná analýza přeskočena. Řešení soustavy s parametrem komplexní frekvence, které je v projektu plně implementované, má srovnatelný výkon s řešením komplexní soustavy pro jednu frekvenci.

Nevýhodou řešení soustavy s komplexními čísly je potřeba ji provádět pro každou frekvenci, která se v obvodu vyskytuje. Metoda řešení soustavy s parametrem je sice potřeba provést pouze jednou, ale v některých případech může docházet k takovému nárůstu složitosti zlomku v buňkách matice, že jsou překročeny limity čísel s plovoucí řádovou čárkou.

Parametry výkonnostních testů	
Zahřátí	30 iterací, každá 1 sekundu
Měření	100 iterací, každá 1 sekundu
Větvě	5
Vlákna	1
Režim	průměrný čas běhu

Výkonnostní test	**Skóre**	**Chyba**	**Jednotky**
Řešení obvodu 50 kondenzátorů v sérii s parametrem	3597	± 12	μs/op
Řešení obvodu 100 kondenzátorů v sérii s parametrem	15392	± 36	μs/op
Řešení obvodu 200 kondenzátorů v sérii s parametrem	72238	± 708	μs/op
Řešení obvodu 50 kondenzátorů v sérii komplexními čísly	4282	± 26	μs/op
Řešení obvodu 100 kondenzátorů v sérii komplexními čísly	22425	± 190	μs/op
Řešení obvodu 200 kondenzátorů v sérii komplexními čísly	111856	± 439	μs/op

Tabulka 6: Výkonnostní testy metody řešení obvodů sériově zapojených kondenzátorů s jedním harmonickým zdrojem napětí



(a) Řešení s parametrem

(b) Řešení pomocí komplexních čísel

Obrázek 9: Krabicové grafy výsledků výkonnostních testů metody řešení obvodů sériově zapojených kondenzátorů s jedním harmonickým zdrojem napětí

7 Implementace logických obvodů

Modul pro simulaci logických obvodů je – stejně jako modul pro řešení elektronických obvodů – bez závislostí mimo standardní knihovnu jazyka Java. Modul umožňuje sestavit nejen kombinační, ale i sekvenční logické obvody.

7.1 Datová struktura

Logické obvody lze také reprezentovat grafem, který je tentokrát orientovaný. Orientace hrany vyjadřuje připojení výstupu logického hradla na vstup logického hradla. Jeden výstup může být připojen na více vstupů. Datová struktura nedovoluje zapojit více výstupů na jeden vstup, jedná se totiž o chybné zapojení, které může způsobit zkrat.

Základem je třída `LogicDevice`, která je základem pro logická hradla a ostatní zařízení, které zajišťují vstup a výstup z logických obvodů. Logické zařízení se skládá z logických vstupů a výstupů. Logické výstupy uchovávají svou aktuální hodnotu na výstupu, to je potřeba v sekvenčních obvodech, kde můžou být hradla zapojena se zpětnou vazbou. Na obrázku 10 je třídní diagram hlavních tříd modulu logických obvodů.

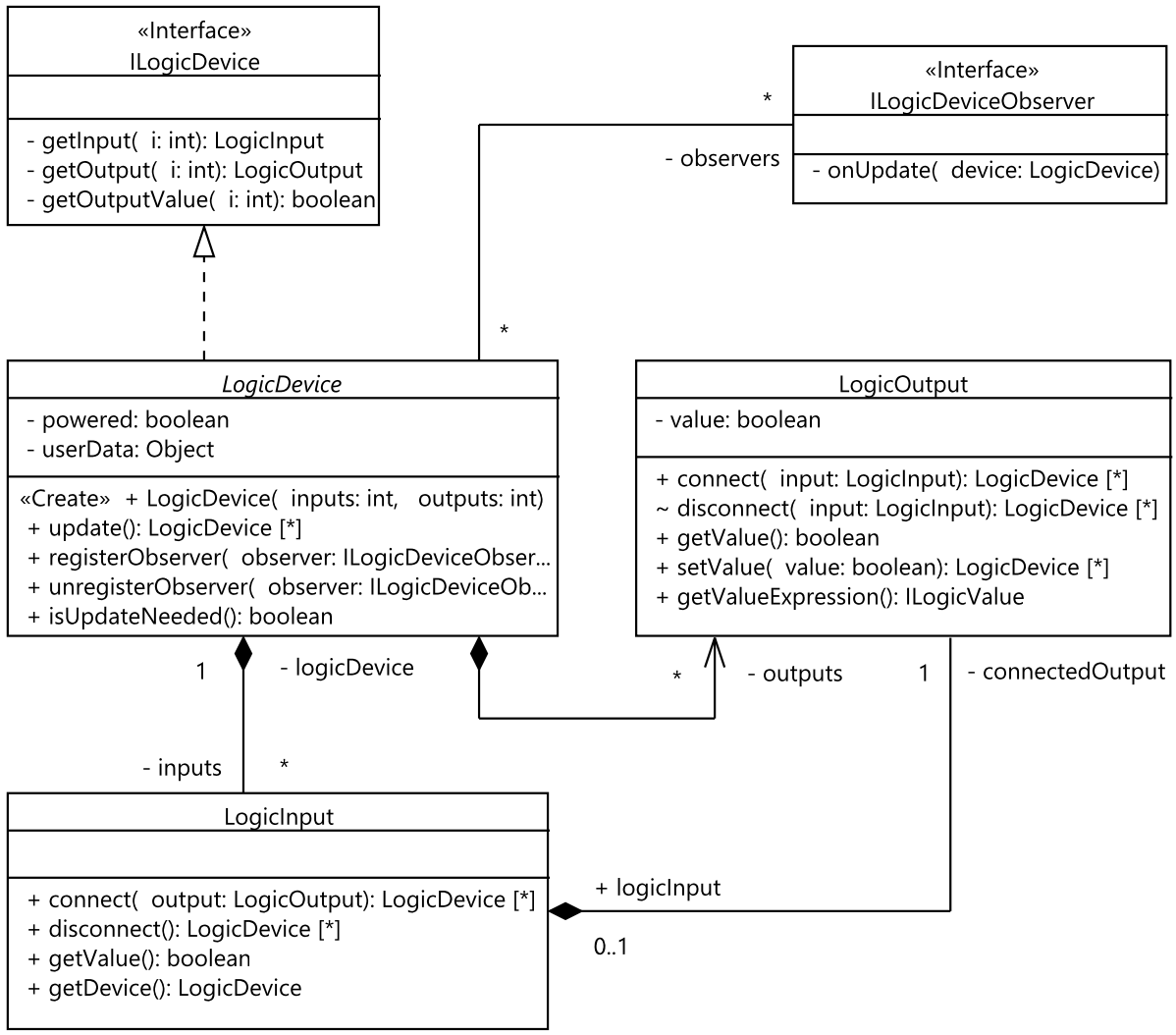
Kromě logických hradel jsou také implementovány prvky, které mají pouze vstup nebo výstup. Prvky, které mají pouze výstup slouží pro vstup uživatele nebo mohou být výstup senzorů. Prvky, které mají pouze vstup, slouží naopak jako výstupu z obvodu, tím může být například světelná indikace logické hodnoty nebo ovládání sepnutí relé.

7.2 Propagace změn

Aktualizace výstupů logických zařízení probíhá změnou logického vstupního zařízení, tím je například přepínač. Nejjednodušší možnost je rekurzivně aktualizovat zařízení, jejichž vstupy jsou připojeny na výstup, jehož hodnota byla změněna. Aby bylo zajištěno zastavení rekurze v obvodech se zpětnou vazbou, je potřeba označit, která zařízení již byla aktualizována. Pořadí aktualizací zařízení v rekurzivním průchodu je ekvivalentní s průchodem grafu do hloubky. Rekurzivní způsob může mít vyšší paměťové nároky, proto byl nahrazen průchodem v cyklu.

Pořadí aktualizací logických zařízení průchodem do hloubky není ideální volbou. Problém může nastat například v logickém obvodu znázorněném na obrázku 11. Pokud je tento obvod aktualizován v pořadí: g_2, g_4, g_1, g_3, pak můžeme vidět v tabulce 7, že při počátečních stavech v čase $t = 0$ a změny hodnoty proměnné A v čase $t = 1$, je potřeba provést dvě aktualizace v pořadí procházení do hloubky, aby výstupy všech hradel byly správné.

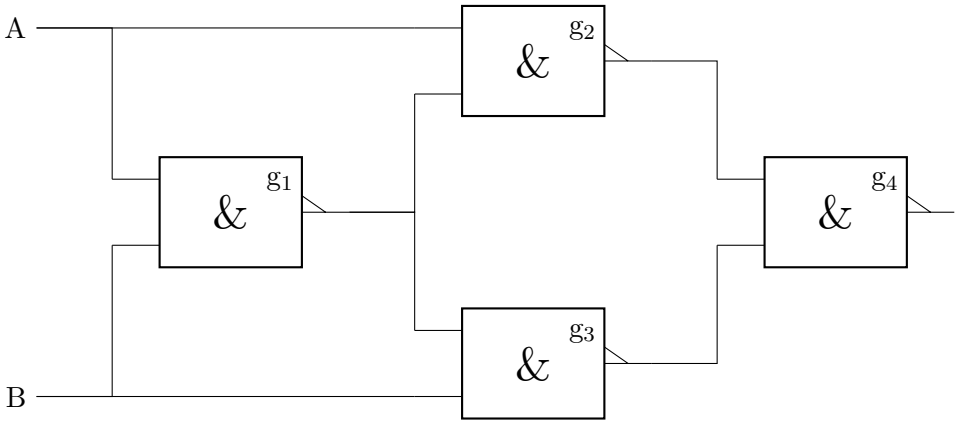
V pořadí aktualizací při procházení do hloubky může docházet k tomu, že výstup hradla se aktualizuje podle výstupů jiných hradel, která ještě nebyla aktualizována. Pořadí aktualizací podle prohledávání do šířky je také nevhodné, protože pokud je opět navštíveno hradlo g_2 jako první, je aktualizováno podle výstupu hradla g_1, které dosud aktualizováno nebylo.



Obrázek 10: Třídní diagram modulu pro simulaci logických obvodů

Pořadí aktualizací hradel musí být takové, aby hradla byla aktualizována dříve, než je jejich hodnota použita pro aktualizaci jiných hradel. To splňuje topologické řazení, které lze jednoduše najít modifikací algoritmu prohledávání do hloubky. Uvažujme variantu algoritmu, kde uzlům jsou přiřazovány stavy neobjeven, objeven, navštíven a dokončen. Uzlům se v tomto algoritmu přiřazuje stav dokončen v opačném topologickém pořadí. Stačí tedy uzel přidat na začátek seznamu uzlů v topologickém pořadí, když je stav uzlu změněn na dokončen. Výpis 2 obsahuje implementaci tohoto algoritmu.

Navíc při objevování uzlů umožňuje algoritmus klasifikovat hrany podle stavu uzlu, který objeví. Pro účely aktualizace stavu logických hradel jsou zajímavé zpětné hrany. Zpětné hrany jsou ty, které vedou z aktuálně procházeného uzlu do jiného již navštíveného, ale nedokončeného uzlu. Pokud je nalezena taková hrana, znamená to, že v grafu je cyklus a tedy logická hradla jsou na sobě cyklicky závislá. Tyto hradla se uloží do dalšího seznamu, aby mohla být po samotné



Obrázek 11: Funkce exkluzivního součtu z hradel negovaného součinu

t	A	B	g_1	g_2	g_3	g_4
0	1	1	0	1	1	0
1	0	1	1	1	0	0
2	0	1	1	1	0	1

Tabulka 7: Stavy proměnných a výstupů hradel v čase, při pořadí aktualizace: g_2, g_4, g_1, g_3

aktualizaci v topologickém pořadí zkontrolována, jestli jejich výstup odpovídá vstupům. Pokud výstup hradla neodpovídá vstupům, nebyl nalezen ustálený stav obvodu a je potřeba na těchto hradlech znovu spustit aktualizaci.

7.3 Sestavení logických funkcí

V logických obvodech je důležité najít logické hodnoty na vstupech do zařízeních, která slouží jako výstup z obvodu. Zařízení, která slouží jako výstup z obvodu, jsou ty, která mají zaregistrovaného pozorovatele sledujícího změnu stavu vstupů zařízení. Stavy hradel obvodu nejsou důležitá, pokud nejsou zapojena tak aby obsahovala zpětnou vazbu. Hodnotu na vstupech výstupních zařízení lze tedy vyjádřit jako logickou funkci, kde jako proměnné budou vystupovat hodnoty na výstupech zařízení, která slouží jako vstup do obvodu.

Logické funkce se skládají z objektů, které implementují funkční rozhraní `ILogicValue`. Logické hradlo aktualizuje logické funkce svých výstupů tak, že vezme logické funkce ze svých vstupů a složí je do nové logické funkce implementující rozhraní `ILogicValue`. Je tedy potřeba, aby se logické funkce sestavovaly v topologickém pořadí. Hodnota vstupních zařízení vystupuje jako proměnná v logických funkcích hradel.

Sestavované logické funkce jsou uchovávány v objektu typu `LogicExpression` a kromě samotné funkce obsahují seznam proměnných, které jsou použity v logické funkci. Pokud je na logickém zařízení zaregistrován pozorovatel, zařízení zaregistruje pozorovatele změny logické hodnoty na všechny proměnné logických funkcí svých výstupů.

```
public static Collection<LogicDevice> update(Collection<LogicDevice> devices) {
     LinkedList<LogicDevice> stack, topologicalOrder, cyclicDependencies; // ...
     Object token = new Object();
     stack.addAll(devices);
     for(LogicDevice dev: devices)
        dev.state = State.DISCOVERED;
     stackLoop: while(!stack.isEmpty()) {
        LogicDevice device = stack.peekFirst();
        switch(device.state) {
           case DISCOVERED:
              device.state = State.VISITED;
           case VISITED:
              for(LogicOutput output: device.outputs) {
                 for (LogicInput input : output.getConnectedInputs()) {
                    LogicDevice nextDev = input.getDevice();
                    if(nextDev.updateIdempotency != token) {
                       nextDev.updateIdempotency = token;
                       nextDev.state = State.DISCOVERED;
                       stack.addFirst(nextDev);
                       continue stackLoop;
                    } else switch(nextDev.state){
                       case VISITED: // Back edge
                          cyclicDependencies.add(nextDev);
                          break;
                    }
                 }
              }
              device.state = State.FINISHED;
              topologicalOrder.addFirst(stack.pollFirst());
              break;
           case FINISHED:
              stack.pollFirst();
              break;
        }
     }
     for(LogicDevice dev: topologicalOrder)
        dev.updateOutputs();
     for (Iterator<LogicDevice> i = cyclicDependencies.iterator(); i.hasNext();) {
        LogicDevice dev = i.next();
        if(!dev.isUpdateNeeded())
           i.remove();
     }
     return cyclicDependencies;
  }
```

Výpis 2: Aktualizace logických zařízení v topologickém pořadí (části kódu jsou vynechány)

K sestavení logických funkcí lze použít stejný algoritmus jako pro aktualizaci logických hodnot. Navíc, pokud je nalezena zpětná hrana, je zařízení označeno, že má cyklickou závislost. Výstup z hradel, která mají cyklické závislosti, se v logických funkcích výstupů ostatních hradel objevuje jako další proměnná.

Když jsou takto sestaveny logické funkce pro všechny výstupy obvodu, stačí při změně vstupu pouze přepočítat výstupy hradel, která jsou potřeba. Na používaných funkcích lze navíc provést minimalizaci.

7.4 Minimalizace logických funkcí

Logické funkce na výstupech hradel mají stromovou strukturu, která může mít mnoho úrovní. Minimalizace logické funkce do jedné metody může snížit časovou i paměťovou náročnost výpočtu hodnoty funkce.

Pro minimalizaci logických funkcí je implementován algoritmus podle Quineovy–McCluskeyho metody popsané v sekci 4.1. Implikanty jsou reprezentovány binárně ve dvou celočíselných hodnotách. Bity jedné hodnoty určují samotné logické hodnoty jednotlivých proměnných implikantu. Druhá hodnota značí pozice, na kterých je pomlčka. Také je implementována Petrickova metoda pro nalezení minimální množiny pokrývajících implikantů, která je popsána v sekci 4.2.

Díky binární reprezentaci lze většinu operací s implikanty možné provádět pomocí binárních operací bez procházení jednotlivých bitů, kromě počátečního rozdělení implikantů podle počtu jedničkových bitů.

Algoritmus na začátku hledá mintermy funkce, pokud počet těchto mintermů přesáhne 2^{n-1}, kde n je počet proměnných logické funkce, algoritmus vytvoří negaci vstupní funkce a hledá minimalizovanou podobu této negované funkce. To v některých případech jednoduše sníží počet implikantů, bez toho aby musel být algoritmus přizpůsoben pro hledání minimální funkce ve tvaru konjunktivní normální formy.

Binární reprezentace omezuje maximální počet proměnných minimalizované funkce. Počet implikantů může být až 2^{n-1}. Implementace algoritmu je použitelná pokud je počet proměnných nejvýše šest. Minimalizace logických funkcí s více než šesti proměnnými je značně časově i paměťově náročnější.

7.4.1 Výkonnostní test minimalizace logických funkcí

V tabulce 8 jsou výsledky výkonnostních testů aktualizace logických výstupů testovacího logického obvodu. Testovací logický obvod obsahuje 5 vstupních zařízení, 3 výstupní zařízení a 9 hradel. Všechny výkonnostní testy mají zaregistrované pozorovatele na koncových výstupech obvodu a mění stejné vybrané vstupy. Výkonnostní testy využívající logické funkce, mají na začátku již tyto funkce sestavené. Pro srovnání je v tabulce uveden i výkonnostní test stejného logického obvodu, ale simulovaného pomocí simulace elektrických obvodů.

Parametry výkonnostních testů	
Zahřátí	30 iterací, každá 1 sekundu
Měření	100 iterací, každá 1 sekundu
Větvě	10
Vlákna	1
Režim	průměrný čas běhu

Výkonnostní test	Skóre	Chyba	Jednotky
Aktualizace všech výstupů v topologickém pořadí	10812	± 28	ns/op
Aktualizace koncových výst. pomocí log. funkce	650	± 2	ns/op
Aktualizace koncových výst. pomocí minimalizované log. f.	337	± 1	ns/op
Minimalizace koncových logických funkcí	7953	± 21	ns/op
Aktualizace log. obvodu pomocí simulace el. obvodů	666265	± 1432	ns/op

Tabulka 8: Výkonnostní testy aktualizace logických výstupů

V tabulce 9 jsou výsledky výkonnostních testů minimalizace funkcí s různým počtem proměnných. Funkce jsou generovány náhodně, ale vždy tak aby byly složeny z 2^{n-1} mintermů. Měří se tedy minimalizace funkcí s maximálním počtem mintermů, protože pokud by jich bylo více, provede se minimalizace funkce opačné.

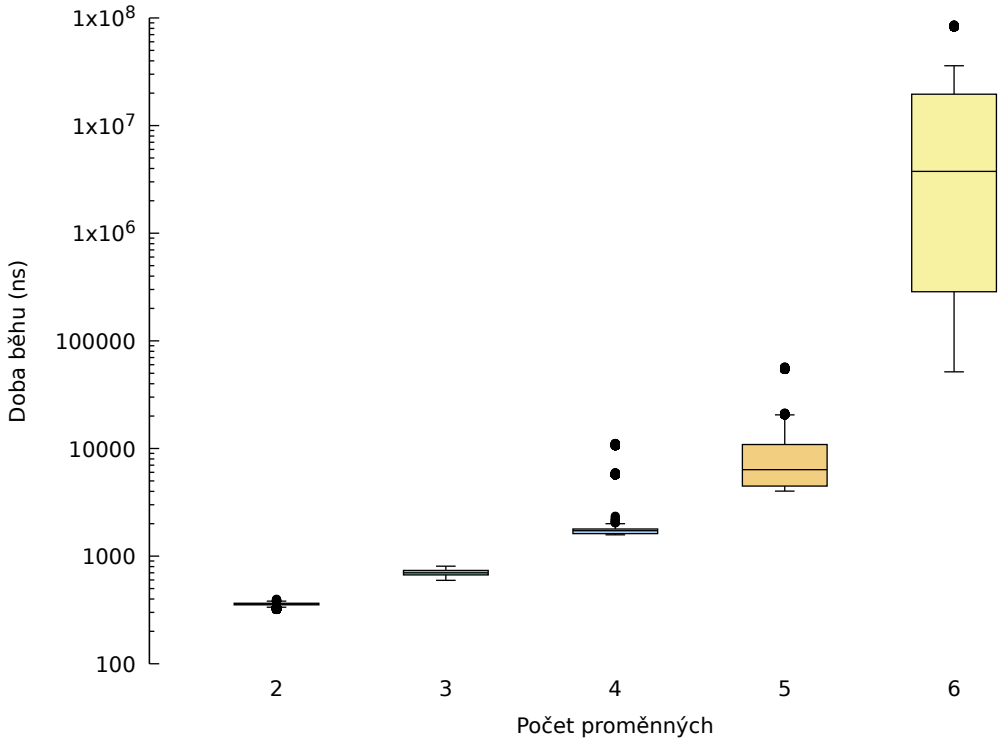
Na krabicovém grafu na obrázku 12 lze vidět, že čas potřebný na minimalizaci funkce roste exponenciálně do pěti proměnných. Minimalizace funkce se šesti proměnnými trvá několik sekund. Paměťová náročnost nebyla měřena, ale při pokusu o minimalizaci funkce se sedmi proměnnými došlo často k selhání, kvůli nedostatku operační paměti.

Ve výsledcích výkonnostních testů můžeme vidět v tabulce 8, že velké zlepšení přináší vyjádření výstupů obvodu pomocí logické funkce. Minimalizace této logické funkce již nepřináší tak velké zrychlení a při vyšším počtu proměnných je minimalizace příliš pomalá.

Parametry výkonnostních testů	
Zahřátí	30 iterací, každá 1 sekundu
Měření	100 iterací, každá 1 sekundu
Větvě	10
Vlákna	1
Režim	průměrný čas běhu

Výkonnostní test	Skóre	Chyba	Jednotky
Minimalizace funkce 2 proměnných	355	± 2	ns/op
Minimalizace funkce 3 proměnných	698	± 5	ns/op
Minimalizace funkce 4 proměnných	2998	± 298	ns/op
Minimalizace funkce 5 proměnných	12664	± 1552	ns/op
Minimalizace funkce 6 proměnných	15396332	± 2620800	ns/op

Tabulka 9: Výkonnostní testy minimalizace logické funkce metodou Quine McCluskey s různým počtem proměnných



Obrázek 12: Krabicový graf výsledků výkonnostních testů minimalizace logické funkce metodou Quine McCluskey

7.5 Serializace a deserializace

Z kontextu obvodu lze získat objekt typu `CircuitContextSnapshot`, který obsahuje pouze nezbytné informace k reprezentaci obvodu. Ten lze serializovat pomocí standardní serializace jazyka Java nebo jiných knihoven, které provádějí serializaci a deserializaci reflexivně. Součástky musí správně použít klíčové slovo `transient` jazyka Java, kterým označí všechny své vlastnosti, které se neserializují. Vývody součástek se neserializují a jsou proto `transient`. Informace o zapojení vývodů jsou serializovány s uzly obvodu.

Standardní deserializace jazyka Java vytváří objekty bez zavolání konstruktoru. Deserializované součástky jsou proto v zakázaném stavu, protože se nevytvoří jejich vývody. Proto všechny serializovatelné součástky musí mít správně implementovaný kopírovací konstruktor, kterým se vytvoří kopie deserializované součástky, která by již měla být v použitelném stavu. Kopie se vytváří při sestavení obvodu z `CircuitContextSnapshot` zavoláním metody `construct`. Součástky obsahují metodu `copy`, která reflexivně najde kopírovací konstruktor a vytvoří pomocí něj kopii. Pokud kopírovací konstruktor chybí, vyhazuje výjimku.

Součástky, uzly i kontexty obvodů umožňují k sobě uložit uživatelská data pomocí metod `setUserData` a `getUserData`. To umožňuje uživateli přidat libovolná data, která mohou sloužit k identifikaci nebo propojení s jinými objekty. Tyto uživatelská data jsou také serializována a musí tedy být serializovatelné.

8 Integrace do projektu Jiný Kosmos

Do hry byly přidány nové bloky, které zastupují prvky obvodu, a simulace, které slouží k sestavování obvodu a samotné simulaci obvodu. K bloku musí být tedy uložena informace odkazující se na prvek obvodu, který zastupuje. Simulace zajišťují provázání mezi herním světem a elektrickými obvody.

Bloky jsou ve hře navrženy tak, že existuje jedna instance třídy popisující blok. Každý typ bloku má unikátní identifikátor, pomocí kterého se na něj lze odkázat. Pokud je potřeba ke konkrétnímu bloku ve světe uložit další informace, lze ke konkrétní pozici ve světě uložit instanci třídy `BlockState`, která ukládá informace způsobem klíč–hodnota.

Nejdříve byly do `BlockState` ukládaly reference na části obvodu. `BlockState` je ale navržen pro hodnotové datové typy a předpokládá se, že stav ovlivňuje vizuální podobu bloku a spouští tedy aktualizaci sloupce světa. Proto byly v `BlockState` ponechány pouze informace ovlivňující vizuální reprezentaci bloku, tím je například stav přepínače. Odkazy na obvod byly přesunuty do nového typu úložiště určeného pro data simulací světa. V projektu je také plánováno rozšíření o síťovou hru více hráčů, kde může být výhodné oddělit běh simulací a jejich data, aby se vyskytovaly pouze na serveru a klienti dostávali jen nezbytné informace o světě.

8.1 Orientace bloků

Aby bylo sestavování složitějších obvodů z bloků pohodlnější, byla do projektu implementována možnost pokládat do světa bloky s různou prostorovou orientací.

Orientace bloku je jednoduše vyjádřena pomocí dvou směrových vektorů. Prvním je normála vrchní strany bloky a druhou je normála přední strany bloku. Z těchto dvou vektorů lze jednoduše získat třetí kolmý vektor a z nich sestavit matici, pomocí které se transformují pozice vrcholů bloku a normály jeho ploch.

Směrové vektory orientace musí být kolmé k některé rovině prostoru. Takových vektorů je pouze šest. Do `BlockState` bloku se ukládá jeden bajt, kde dvě trojice bitů udávají pořadí prvku z výčtového typu `Position.Direction`, ze kterého se dá získat samotný vektor. Důvodem je úspora paměti, neboť různá orientace bloků by mohla být v budoucnu použita častěji, třeba i generátorem světa.

Do třídy `Block`, ze které dědí všechny bloky, byly přidány dvě booleovské vlastnosti. První nastavuje jestli blok používá orientaci a druhá omezuje orientaci tak, že vrchní strana bloku je vždy nahoře a blok může tedy být otočen pouze kolem jedné osy do čtyř směrů.

Pokládání bloku s možností orientace probíhá tak, že v okamžiku položení bloku se vezme normála strany bloku, na který hráč nový blok pokládá a vektor od místa pokládání k hráči. Normála strany se použije jako první vektor v dvojici vektorů vyjadřující orientaci bloku, tedy směr normály vrchní strany pokládaného bloku. Pokud je u bloku nastavena vlastnost, že je vrchní strana bloku vždy nahoře, je tento vektor nahrazen vektorem se směrem nahoru. Vektor

směrující k hráči se upraví tak, aby byl kolmý k některé rovině prostoru a použije se jako vektor normály přední strany pokládaného bloku.

8.2 Úložiště simulačních stavů

Podobně jako stavy bloků se i simulační stavy ukládají ke sloupci světa (instance třídy `Chunk`). K jedné pozici světa může být uloženo více simulačních stavu, ale každý stav musí patřit k jedné simulační třídě. Pro jednu simulační třídu je na jedné pozici ve světě uložen jeden stav, který ukládán jako reference na `Object`. Simulační třídy si tedy mohou uložit k pozici ve světě objekt libovolného datového typu, ale musí provádět typovou kontrolu při získání tohoto objektu.

Simulace mohou také ukládat stav do globálního úložiště, které je k dispozici vždy, sloupce mohou být totiž odloženy na disk, pokud je hráč od nich daleko. Globální úložiště je v instanci třídy světa `World` a k jedné simulační třídě přiřazuje jednu referenci na objekt.

Objekty uložené v simulačním úložišti musí být serializovatelné. Není vyžadována implementace rozhraní `Serializable`, ale nesmí obsahovat cyklické reference, nebo musí být vhodně použito klíčové slovo `transient`. K serializaci se používá JSON serializace z rámce libGDX, takže je možné provést vlastní serializaci, pokud stavový objekt implementuje `Json.Serializable`.

Některé simulace může být potřeba spustit po načtení. K tomu slouží anotace `@RunOnLoad`, kterou lze označit statickou metodu simulační třídy. Takto označená metoda se zavolá, pokud byl načten simulační stav pro simulační třídu, která obsahuje označenou metodu. Zavolá se v případě načtení globálního simulačního stavu, ke kterému dojde při načtení světa, ale také při načtení simulačního stavu bloku, který je načten spolu se sloupcem. Metoda označená anotací `@RunOnLoad` může mít parametry, které musí být typu `World`, `SimulationManager`, `Position` nebo typ načteného simulačního stavu.

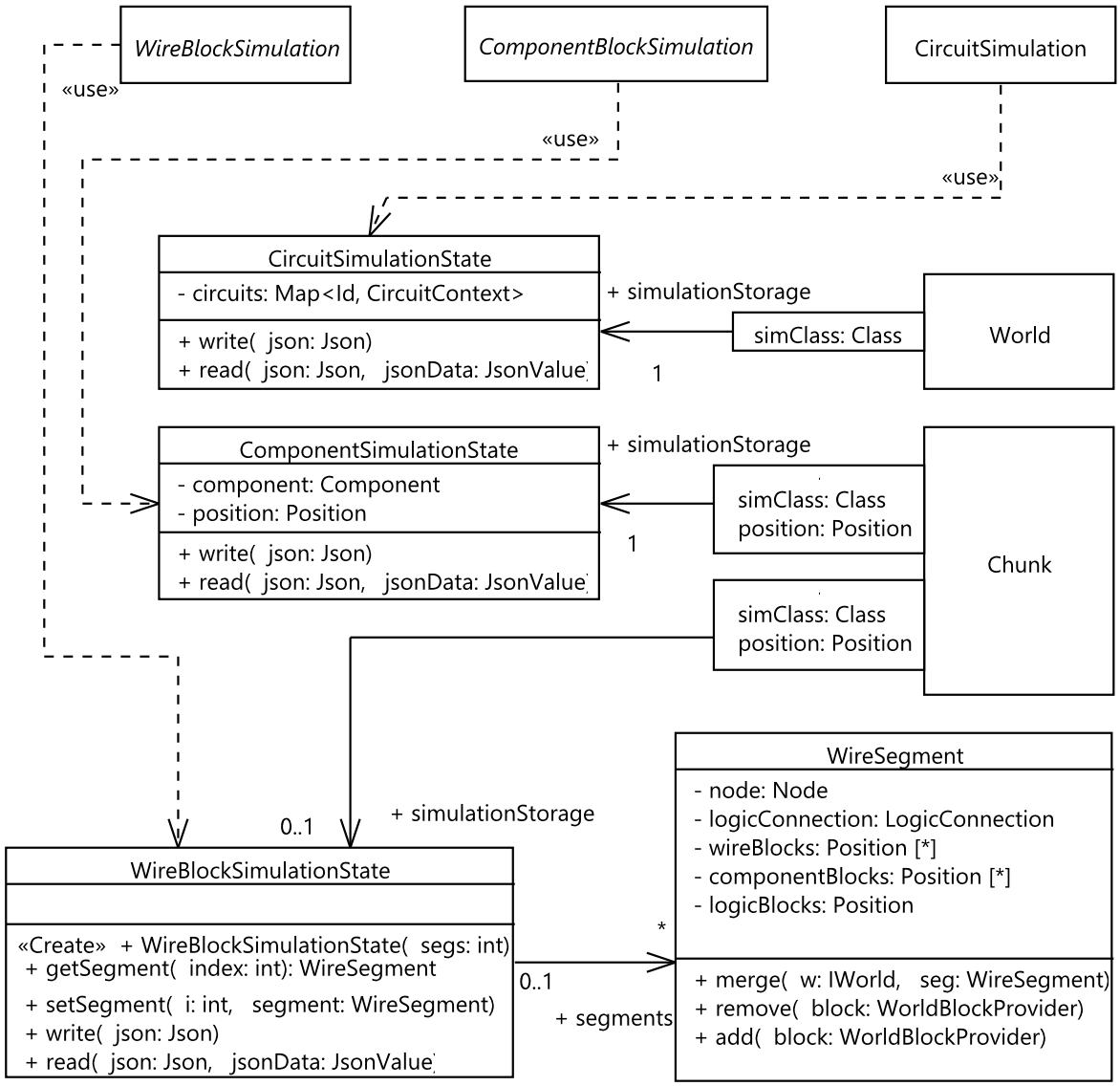
Obrázek 13 vizualizuje, jak simulační třídy používají simulační úložiště. Tyto třídy jsou popsány v následujících sekcích.

8.3 Vodiče

Jeden vodič se může skládat z více bloků, ale celý vodič zastupuje jeden uzel obvodu. Bloky vodičů proto mají v simulačním úložišti objekt, který popisuje celý segment vodiče. Bloky, které jsou navzájem propojené, mají stejný objekt segmentu vodiče.

Bloky vodičů implementují rozhraní `IWireBlock`, pomocí kterého určí kolik segmentů vodičů blok zastupuje a na kterých stranách bloku jsou vyvedeny. Blok vodiče může být tedy složen z více segmentů vodičů a na jedné straně bloku může být vyvedeno více segmentů. Pokud je na jedné straně vyvedeno více segmentů, je důležité jejich pořadí. Toho využívá blok pro distribuci napájení, který přes jednu stranu bloku poskytuje dva oddělené vodiče.

Třída `WireSegment` popisuje segment vodiče a obsahuje uzel obvodu, pozice bloků vodiče a ostatních připojených bloků. O vytvoření instance třídy segmentu vodiče se stará simulace typu `WireBlockSimulation`. Ta má dva podtypy, jeden pro přidání bloku a druhý pro odstranění



Obrázek 13: Třídní diagram simulačních stavů simulací pro řešení obvodu

bloku. Simulace se tedy spouští pokud se blok položí nebo odstraní. Segmenty vodičů jsou zaobaleny třídou `WireBlockSimulationState`, která je vložena do simulačního úložiště bloku. Každý blok vodiče má vlastní instanci třídy simulačního stavu bloku vodiče. Obsažené instance třídy popisující segment vodiče mohou bloky sdílet, pokud jsou ve stejném segmentu.

Pokud je blok vodiče položen, simulace zkontroluje okolní bloky. Pokud je vedle aktuálního bloku položen jiný blok vodiče, který má na dotýkající se straně vyveden segment vodiče, aktuální blok se přidá do nalezeného segmentu. Simulace dále pokračuje v kontrole ostatních sousedních bloků a pokud nalezne další blok s jiným segmentem, provede sloučení obou segmentů. Pokud není nalezen žádný sousední připojený segment vodiče, vytvoří se nový segment obsahující pouze aktuální blok.

Když je blok odstraněn, může to znamenat rozdělení segmentu vodiče na dva samostatné vodiče. Simulace proto odstraní segment ze všech jeho bloků a spustí stejnou metodu pro vytvoření segmentu, která je použita při položení bloku, na všech blocích kromě odstraněného.

V projektu jsou implementovány dva typy bloků vodičů. První blok typu `WireBlock` propojuje všechny strany bloku a slouží jako hlavní propojovací blok. Druhým je blok typu `PowerDistributionBlock`, který má dva segmenty vodičů. Tyto segmenty jsou překřížené, ale nejsou spojené. Jeden segment spojuje severní a jižní stranu bloku, druhý západní a východní. Oba segmenty jsou vyvedeny do horní strany bloku a poskytuje tak napájení pro podporované bloky. Pomocí tohoto bloku lze postavit napájecí platformu pro logická hradla, nebo jiná zařízení, která potřebují napájení.

8.4 Bloky součástek obvodu

Bloky zastupující součástky obvodu jsou navrženy tak, že jeden blok zastupuje jen jednu součástku obvodu a její vývody jsou vyvedeny na nějaké strany bloku. Bloky implementují rozhraní `IComponentBlock`, kterým určí zastupovanou součástku a strany vývodů. Podobně jako bloky vodičů mohou mít na jedné straně bloku více segmentů, může i blok součástky mít na jedné straně více vývodů.

Vytvoření součástky a její propojení s okolními bloky nebo odstranění a odpojení součástky zajišťuje simulace typu `ComponentBlockSimulation`, která má dva podtypy: `Update` pro aktualizaci zapojení a `Remove` pro odstranění všech zapojení bloku. Aktualizace zapojení se spouští, pokud je blok položen. Také může být spuštěna ze simulace bloku vodiče, když je položen vodič vedle bloku součástky nebo pokud je segment sousedního vodiče změněn.

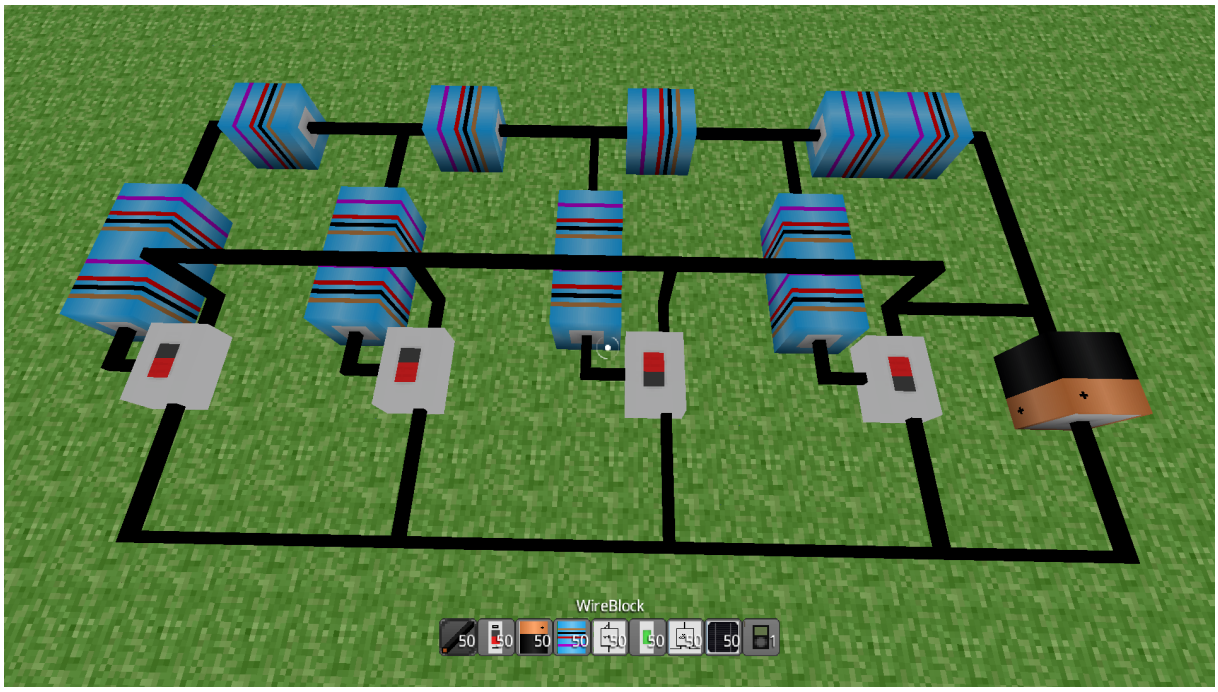
Většina bloků součástek má podobnou implementaci rozhraní `IComponentBlock`. Toto společné chování bylo přesunuto do abstraktního bloku `SimpleComponentBlock`. Ten obsahuje mapu strany bloku na index vývodu součástky a implementuje rozhraní `IComponentBlock` tak, aby vracel správný vývod s ohledem na orientaci bloku.

Reference na součástku je obsažena v simulačním stavu simulace součástek (instance třídy `ComponentSimulationState`), který je vložen do simulačního úložiště bloku.

V projektu jsou implementovány bloky základních součástek jakým jsou rezistor, cívka, kondenzátor a přepínač. Jako zdroje energie slouží bloky zastupující baterii, solární panel nebo třífázový generátor. Hra zatím neobsahuje denní cyklus nebo mraky, proto je solární panel zatím jen zdroj kolísajícího napětí. Generátor má sloužit k přeměně točivé energie na elektrickou, ale do hry zatím nebyl přidán žádný zdroj točivé energie a proto aktuálně generuje elektrické napětí, i když není na jeho rotor nic připojeno. Další bloky součástek jsou elektrická pec, světelný indikátor průchodu elektrické proudu a usměrňovací dioda.

Příklad obvodu sestaveného z bloků je na obrázku 14, jedná se o rezistorovou síť R-2R, která se používá v D/A převodnících.

Hodnota veličiny součástky bloku lze změnit pomocí nástroje na změnu nastavitelných veličin. Po kliknutí na součástku s tímto nástrojem, je zobrazeno okno s posuvníkem, kterým lze hodnotu součástky měnit. Minimum, maximum a krok posuvníku udává blok součástky.



Obrázek 14: Odporová síť R-2R

8.5 Bloky logických zařízení

Bloky logických hradel a ostatních logických zařízení mají na svých stranách logické vstupy a výstupy. Na spodní straně bloku je dvouvodičové připojení k napájení. Bloky lze připojit k napájení položením na napájecí platformu. Součástí bloku je také rezistor, který je zapojen na napájení a slouží jako umělá zátěž. Blok naslouchá změně obvodu rezistoru a mění stav napájení logického zařízení podle úrovně napětí. Toto napájení slouží pouze proto, aby byl hráč nucen

logická zařízení napájet. Nebyly řešeny nesprávné zapojení napájení, proto například propojení dvou hradel, která jsou odděleně napájena bez společné země, bude funkční.

Při položení nebo odstranění bloku logického zařízení je spuštěn příslušný typ simulace `ComponentBlockSimulation`, která se stará o zapojení zátěžového rezistoru. O zapojení samotného logického zařízení se stará simulace `LogicBlockSimulation`, která je velmi podobná simulaci bloků komponent. Zapojení nebo odpojení logického zařízení rovnou spouští aktualizaci zařízení, přitom může nastat situace, kdy jsou nalezeny cyklické závislosti. V tom případě vrací metody, které slouží k připojování nebo odpojování, kolekci zařízení s cyklickými závislostmi vyžadující aktualizaci. Na každém zařízení, které potřebuje aktualizaci, se spustí nová simulace `LogicDeviceSimulation`, která provádí pouze aktualizaci logického zařízení a nekontroluje jeho zapojení. Tato simulace se provádí opakovaně, pokud zařízení stále potřebuje aktualizaci. Díky tomu se může logický obvod rozkmitat.

V projektu jsou bloky logických hradel AND, OR, NOT a XOR. Blok spínače má variantu, která přepíná mezi logickými hodnotami. Taktéž blok indikátoru průchodu elektrického proudu má variantu, která indikuje logickou hodnotu na vstupu. Tyto speciální varianty bloků jsou použity, pokud jsou tyto bloky položeny na napájecí platformu.

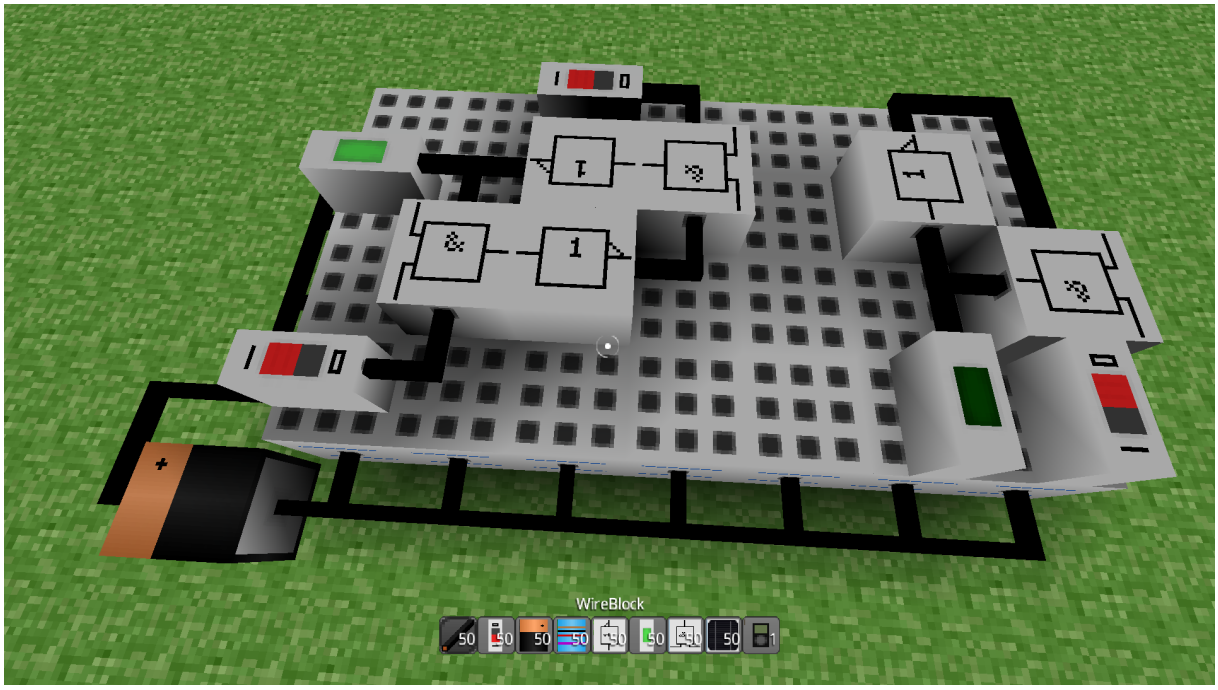
Snímek zapojení RS klopného obvodu a jednoduchého oscilujícího logického obvodu je na obrázku 15. Zapojení složitějšího obvodu lze vidět na obrázku 16, kde je zapojení 3-bitové sčítačky složené ze tří úplných sčítaček.

Do projektu byly také přidány bloky, pomocí kterých lze propojit logické a elektrické obvody. Jedním z nich je relé, jehož spínání je ovládáno logickým obvodem. Druhým blokem je operační zesilovač, který nastavuje svůj výstup na logickou 1 nebo 0 podle rozdílu napětí na vstupech.

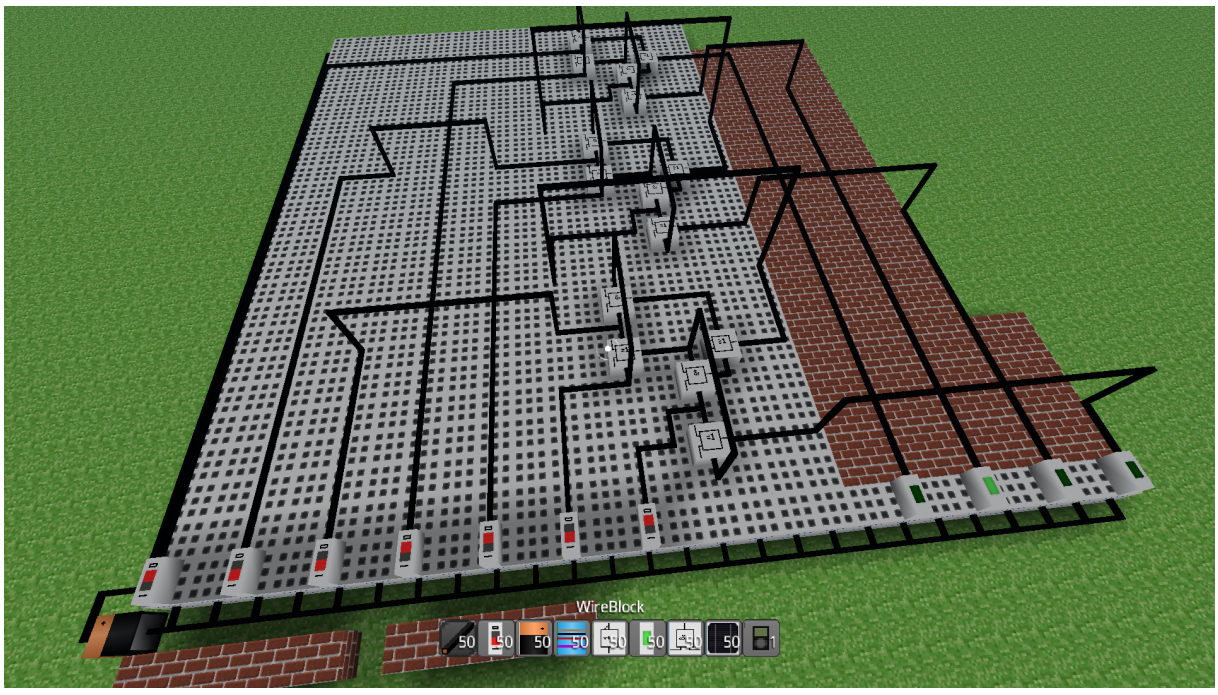
8.6 CircuitSimulation

Simulace hry jsou navrženy tak, aby byly spojeny s jedním blokem na nějaké pozici ve světě a aby pro každý blok mohla existovat nejvýše jedna simulace. Simulace obvodu je speciální v tom, že může být nejvýše jedna pro jeden obvod. Nemůže se tedy stát, že by jeden obvod řešilo více simulací z různých bloků. Tato simulace nepracuje se samotným obvodem, ale s celým kontextem obvodu, který byl popsán v sekci 6.5.

Simulace spustí analýzu obvodů kontextu, pokud je potřeba, a aktualizaci součástek o dobu od posledního simulačního kroku. Simulace zjišťuje doporučený krok aktualizace součástek a provede aktualizaci součástek opakovaně, pokud je simulační krok větší než doporučený. Pokud obvod nepotřebuje znovu provést analýzu nebo v poslední aktualizaci součástek nebyla žádná aktualizována, simulace se ukončí. V opačném případě se simulace ukončí se stavem `CONTINUE` a bude pokračovat v dalším simulačním kroku.



Obrázek 15: RS klopný obvod a jednoduchý oscilující logický obvod



Obrázek 16: 3-bitová sčítačka

8.7 Multimetr

Hráč může pomocí multimetru sledovat chování obvodu. Jedná se o herní předmět, který dokáže měřit napětí a proud. Kliknutím na dva různé vodiče, lze mezi nimi měřit napětí. Kladný vývod multimetru lze umístit levým tlačítkem myši a záporný pravým tlačítkem myši. Na levé straně obrazovky zobrazuje hodnoty a průběh veličin v čase. Rychlost vzorkování lze změnit pomocí klávesy „shift“ a kolečka myši. Pomocí klávesy „T“ lze zapnout spouštěč, který způsobí, že se průběh začne vykreslovat až narazí na náběžnou hranu přes hledanou hodnotu. To je užitečné při měření harmonického napětí. Hledaná hodnota spouštěče lze změnit pomocí kombinace klávesy „control“ a kolečka myši.

Kliknutím multimetrem na jednobran lze měřit nejen napětí mezi jeho vývody, ale také proud jím procházející.

Funkčnost multimetru je rozdělena do tří tříd. První je samotný herní předmět, který může mít hráč v inventáři. Do hry byla přidána možnost interaktivních předmětů, které implementují rozhraní `IIteractableItem`. To umožňuje předmětu sledovat vstupy uživatele a také interagovat s bloky ve světě.

O zobrazení průběhu veličin se stará třída `MultimeterActor`, což je prvek uživatelského rozhraní. Pokud je multimetr vybrán, spustí se simulace `MultimeterSimulation`. Ta vzorkuje měřené veličiny a posílá je objektu, který zobrazuje veličiny na uživatelském rozhraní.

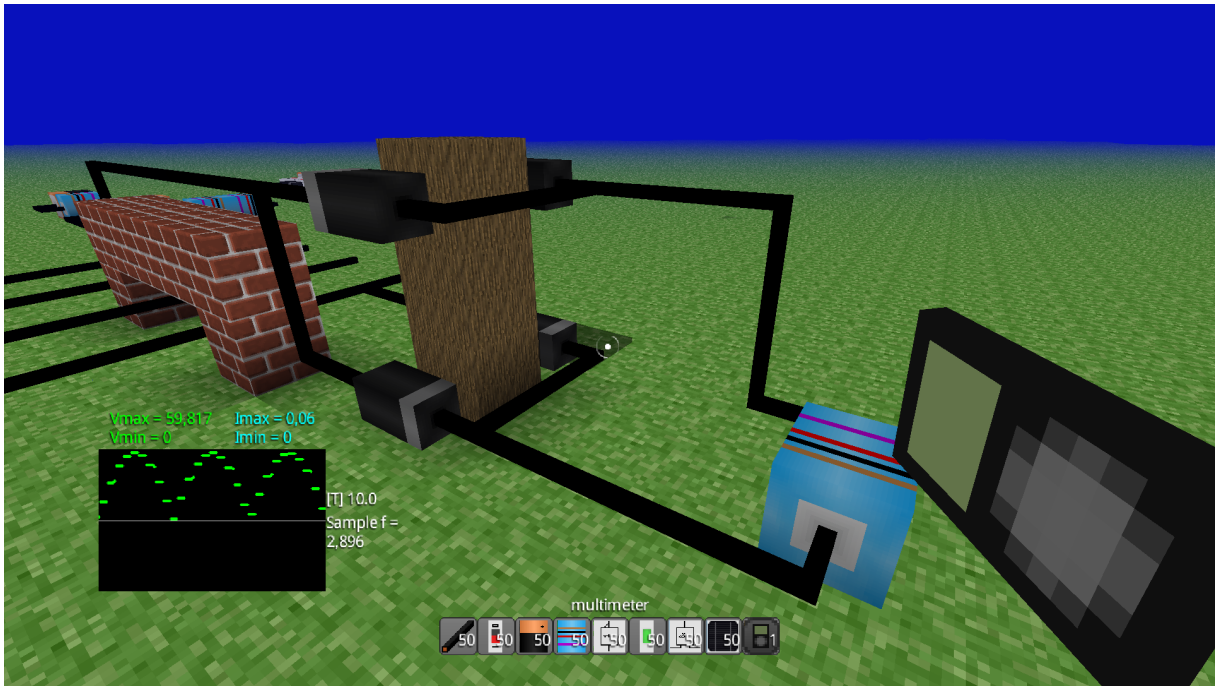
Rychlost vzorkování může být rychlejší než rychlost aktualizace měřeného obvodu. V případě stejnosměrných veličin to nemá význam, protože mezi aktualizacemi obvodu se veličiny nemění. Pokud obvod obsahuje harmonické zdroje, pak se zobrazuje správná hodnota těchto harmonických veličin i mezi aktualizacemi obvodu. Harmonické veličiny jsou vyjádřeny tak, že lze zjistit jejich okamžité hodnoty v libovolném čase.

Na obrázku 17 je vidět měření napětí na rezistoru, který je připojen k usměrňovacímu můstku. Průběh napětí je schodovitý, protože usměrněné harmonické napětí se mění pouze po aktualizaci obvodu. Vertikální měřítko grafu průběhu veličin je automaticky upravováno tak, aby byl využit celý prostor a aby nebyly žádné hodnoty mimo vykreslovaný graf.

8.8 Ukládání

Elektrické obvody lze snadno serializovat, to je popsáno v sekci 7.5. Samotná reference na kontext obvodu není nikde jednoduše dostupná, ale bloky součástek mají uloženou referenci na součástku a bloky vodičů na uzel. Tyto reference jsou uloženy v simulačním úložišti. Když se ukládá sloupec světa, ukládají se i simulační úložiště bloků v ukládaném sloupci.

Pro ukládání simulačního úložiště bloku součástky nebo vodiče je využita možnost vlastní serializace, kterou nabízí rámec libGDX. Nejprve se zkontroluje jestli obvod ukládaného prvku je uložen v globálním simulačním úložišti. Pokud není, tak je mu přidělen unikátní identifikátor a je do globálního úložiště uložen. Unikátní identifikátor se skládá z pozice ve světě a indexu. Jako pozice je použita pozice první ukládané součástky nebo vodiče obvodu. Index je důležitý



Obrázek 17: Měření napětí na rezistoru, který je připojen k usměrňovacímu můstku, pomocí multimetru

jen pro případ kdy je ukládán blok, který se skládá z dvou nepropojených vodičů, které jsou v různých obvodech. Pokud se jedná o první ukládaný blok obou obvodů, budou mít oba obvody stejnou pozici. Proto je k identifikátoru obvodu přidán také index segmentu, aby je bylo možno odlišit. Unikátní identifikátor obvodu je vložen do uživatelských dat obvodu.

Součástka bude uložena spolu s celým obvodem v globálním simulačním úložišti. Není tedy třeba ukládat součástku v simulačním úložišti bloku. Místo toho je potřeba uložit odkaz na obvod a jeho součástku. Proto se ukládá pouze identifikátor obvodu a identifikátor součástky. Identifikátorem součástky je opět pozice, která je uložena do uživatelských dat součástky. Na jedné pozici může být pouze jedna součástka a není tedy potřeba přidávat k identifikátoru index.

V případě vodičů jsou do uživatelských dat uzlu uloženy informace o celém segmentu vodiče. Součástí je identifikátor segmentu, kterým je pozice a index.

Uložené sloupce světa tedy obsahují v simulačním úložišti pouze odkaz na obvod a jeho prvek. Po uložení všech sloupců je uloženo globální simulační úložiště do samostatného souboru.

Načítání probíhá v opačném pořadí, nejprve je načteno globální simulační úložiště, které obsahuje kontexty obvodů. Když je poté načten sloupek světa, pomocí odkazů v uloženém simulačním úložišti bloku je dohledána konkrétní součástka nebo segment vodiče z globálního simulačního úložiště. Kontext obvodu je tedy načten, i když nejsou načteny žádné jeho bloky. Simulace obvodu může běžet i pokud je obvod v nenačtené části světa.

Logické prvky jsou uloženy v simulačním úložišti bloku a jsou serializovány tak, že obsahují pouze hodnoty svých výstupů, ale žádné informace o svých vstupech. Logické prvky jsou načteny

z disku až spolu se sloupcem světa. Jejich simulace tedy neběží, pokud logický blok je v nenačteném sloupci na rozdíl od simulace obvodů. Logické bloky jsou po načtení propojeny simulací logických bloků. Během propojování načtených logických bloků jsou aktualizace logických prvků při změně zapojení vypnuty. To zamezí změně hodnot na hradlech se zpětnou vazbou, která by po postupném sestavovaní s aktualizacemi mohla mít jiné hodnoty, než se kterými byla uložena.

9 Závěr

Výsledkem této práce je modul, který dokáže simulovat elektrické a logické obvody. Myslím si, že přesnost simulace je dostačující pro výukové účely. Modul byl integrován do projektu Jiný kosmos. Videohra Jiný kosmos by v této podobě mohla sloužit k podpoře výuky na základních nebo středních školách. Výhodou oproti výuce ve specializované učebně je neomezené množství součástek a možnost experimentovat bez rizika poškození skutečných součástek.

V projektu Jiný kosmos je spousta prostoru pro zlepšení. Pokládání bloků z panelu inventáře na obrazovce by potřebovalo vylepšit, aby bylo možné různé varianty stejného bloku, které by se v případě součástek obvodu mohly lišily hodnotou parametru součástky např.: odpor. Pokládání vodičů může být někdy problematické, protože sousední bloky vodičů jsou propojené a je tedy nutné nechávat mezi vodiči mezery. To by se dalo vyřešit pokládáním vodičů různých barev, kde by pouze vodiče stejné barvy byly propojeny.

Projekt Jiný kosmos je sice v současnosti stále docela strohý, ale věřím, že ho budou další studenti dále rozvíjet.

Literatura

1. BRECHER, Jakub. *Generování map a struktur pro projekt Jiný Kosmos*. 2017. diplomová práce. VŠB – Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, Katedra informatiky.
2. ČECH, Dominik. *Simulace prostředí pro projekt Jiný Kosmos*. 2018. diplomová práce. VŠB – Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, Katedra informatiky.
3. CHEEVER, Erik. *Modified Nodal Analysis - Intro* [online]. Swarthmore College [cit. 2020-02-13]. Dostupné z: `https://lpsa.swarthmore.edu/Systems/Electrical/mna/MNA2.html`.
4. SOUKUP, Lubomír. *Obyčejné diferenciální rovnice a jejich užití ve fyzice*. 2010. bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství.
5. JADHAV, Vitthal; BUCHADE, Amar. *Modified Quine-McCluskey Method*. 2012. Dostupné z arXiv: `1203.2289 [cs.OH]`.
6. *Minimization of Boolean Functions*. 1956-06. Dostupné také z: `https://archive.org/details/bstj35-6-1417`.
7. KRAMBECK, Donald. *Prime Implicant Simplification Using Petrick's Method*. EETech Media, LLC., 2016-02. Dostupné také z: `https://www.allaboutcircuits.com/technical-articles/prime-implicant-simplification-using-petricks-method/`.
8. *Battery (Table-Based)* [online]. The MathWorks, Inc. [cit. 2020-02-13]. Dostupné z: `https://www.mathworks.com/help/physmod/sps/ref/battery.html`.